DOCTOR OF PHILOSOPHY

Using Reinforcement Learning in Solving Exam Timetabling Problems

Han, Kehan

Award date:
2018

Awarding institution:
Queen's University Belfast

USING REINFORCEMENT LEARNING IN SOLVING EXAM TIMETABLING PROBLEMS

by

Kehan Han, M**Sc**

Thesis submitted to

School of Electronic, Electricity Engineering and Computer Science

Queen's University Belfast

For the degree of Doctor of Philosophy

October 2018

ACKNOWLEDGEMENTS

I would like to especially thank my parents and my sister, without whose help and support I would not be here. Also, I would like to thank my supervisor, Dr Paul McMullan, who provided guidance throughout my PhD study. And many thanks to my lovely boyfriend Jonny, who has always been there for me during my last year of PhD.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

There is increasing interest in the area of research involving automated technologies for solving problems. High generality and ease of reusability have become a major goal in the development of Search methodologies within automation. One search method that meets these requirements is the hyper-heuristic. A hyper-heuristic is a unique method as it explores the space of low-level heuristics rather than specific problem domains. Two common types of hyper-heuristic have been proposed in the literature: selection hyper-heuristics and generation hyper-heuristics. In this thesis, the research focus is concentrated on selection hyper-heuristics, using a single point base search in the selection and application of suitable low-level heuristics to the target problem domain during optimization. The selection process includes a learning ability, with the performance of all selected heuristics recorded and updated during the search process.

The learning ability is a major focal point within the study of hyper-heuristic application in this thesis. It is enhanced through introducing Reinforcement Learning algorithms into the design of the proposed single point-based selection hyper-heuristic framework. Various Reinforcement Learning techniques are abstracted and designed as heuristic selection methods and utility update methods. The goal of this part of the research is to create a way to give machine learning more generality and applicability by embedding it into a hyper-heuristic. The learning scheme of the Hyper-heuristic is also studied thoroughly through this multiple component design. Furthermore, a cooperative hyper-heuristic based on previous research results is proposed and implemented. This cooperative framework manages multiple hyper-heuristics with different local settings to perform parallel search to solve optimization problems. Different hyper-heuristic designs have

varying degrees of qualitative performance as target problems change. Rather than aiming to determine a single best hyper-heuristic configuration, the construction of a set of hyper-heuristics with varied parameter settings forms the goal of the learning process. Through the cooperative parallel search, the hyper-heuristic search performance can be improved, allowing a consistent and effective level of generality. The core goal of this research is in achieving this level of generality with the hyper-heuristic approach.

Two general approaches are proposed in particular within this work: Reinforcement Learning Simulated Annealing hyper-heuristic framework and Cooperative hyper-heuristic framework. Both frameworks are implemented and tested using examination timetabling problems as a case study. Examination timetabling is a classic optimization and scheduling problem which has attracted the attention of researchers from various fields. The benchmark selected for the purposes of experimentation and qualitative comparison is the ITC2007 datasets, which closely approximate real-world scheduling problems.

As outlined within this work, the two proposed frameworks have been able to consistently solve all instances of the chosen ITC2007 exam timetabling benchmark datasets, despite the varied characteristics between instances. Furthermore, the proposed cooperative hyper-heuristic managed to deliver competitive results in solving the ITC2007 problem, as compared to current hyper-heuristics currently published in the literature. This work shows that the performance of the hyper-heuristics is improved by applying a cooperative hyper-heuristic approach, while maintaining generality over many instances of the problem. In this research, Reinforcement Learning is embedded into the general hyper-heuristic framework, improving the generality and applicability of machine learning techniques when used to solve complex scheduling and optimisation problems.

CHAPTER 1. INTRODUCTION

1.1 Timetabling and Scheduling Problems

Petrovic and Burke (2004) describes timetabling problems as a certain type of scheduling problem. Although the process of timetabling and scheduling can both be considered forms of resource allocation following certain rules, there is a difference between them. Scheduling problems aiming to minimize the total cost of resources used while timetabling is more focused on the improvement of constraint satisfaction (Wren, 1996). There are disagreements upon this description in the literature. Carter (2001) argues that timetabling decides upon the time when events will take place without involving traditional resource allocation process in scheduling problems.

Timetabling problems are solved by placing events in a certain limited number of time slots and locations subject to constraints. A timetabling solution is evaluated by calculating the extent to which it satisfies these constraints. Timetabling problems exist in a wide variety of areas, including academic institutions (course and exam timetabling), hospitals (nurse rostering), the transport industry (train and bus scheduling) and sport (timetabling of matches between pairs of teams) (Petrovic and Burke, 2004). In the last several decades, research and applications in understanding and solving timetabling problems have taken place at both the academic and real-world level (Petrovic and Burke, 2004.).

This research focuses on exam timetabling problems which mainly arise in universities. Exam timetabling is a difficult combinational problem which is crucial for

most academic institutions as they have to face it every semester. The exam timetabling problem-solving process can be described as placing exams in available time slots and rooms without causing conflicts and trying to satisfy certain constraints as much as possible.

Carter and Laporte et al. (1994) summarizes exam timetabling as a challenging problem. Cooper and Kingston (1995) prove that timetabling is an NP-hard optimization problem. Currently, the growing number of students and courses, let alone the wide range of constraints that must be considered, highlight the challenges involved in exam timetabling at universities. Even small changes in the exams could cause a large movement in the timetable due to the number of people would be affected by the outcome. Therefore, an automated exam timetabling system which features high generality is required.

As one of the most studied educational timetabling problems, researchers from all over the world have been attracted to solve the problem of exam timetabling. In late 1990s, an overview of exam timetabling problems was presented by Burke, Edmund, Jackson et al. (1997) which introduced approaches that were used to solve exam timetabling problems up to the end of the decade. The approach introduced in this overview includes Genetic algorithms (Burke, E. K., Elliman et al. 1995; Weare, Burke et al. 1995; Burke, Edmund K., Newall et al. 1995), Memetic Algorithms (Paechter, Cumming et al. 1995), Simulated Annealing (Thompson et al., 1995, 1996), Tabu search (Hertz, 1991, 1992; Kang et al., 1992) and Constraint logic programming (Boizumault et al., 1994, 1995; Cheng et al., 1995). All these approaches were reported to perform well with the given timetabling datasets.

Petrovic and Burke (2004) provided an extended overview of the exam timetabling problem and proposed to categorize the new, emerging approaches into four main types: (i) metaheuristic approaches, (ii) Multi-criteria approaches, (iii) Case-based reasoning approaches, and (iv) Hyper-heuristic and self-adaptive approaches. This overview also emphasized the future direction of timetabling research, which is towards methods with high generality and less parameter-dependent tuning which can be easily reused when applied to different datasets (Petrovic, Burke 2004). In Qu and Burke et al.'s (2006) survey of exam timetabling methodologies, all the methods that have been used in solving exam timetabling problems and most popular benchmarks are included. Both exact approaches (such as discrete linear-programming and Branch-and-Bound) and non-exact approaches (such as metaheuristic) are introduced to the method's analysis of the performance and limitations of this survey.

In these previous surveys (Burke, Edmund, Jackson et al. 1997; Petrovic, Burke 2004; Qu, Burke et al. 2006), the limitations of most existing methods have been revealed. Although these methods have been able to generate good results on certain datasets, most of these methods are quite problem-specific. The reapplication of these methods to new timetabling problems would always require a great amount of parameter tuning. Research results generated by these approaches could be difficult to cater for different objectives (Burke, Edmund, Jackson et al. 1997) in real-world exam timetabling problems, let alone a wider range of scheduling and optimization problems.

1.2 Scope and Objectives

This thesis focuses on exam timetabling problems. Examination timetabling problems in the real world normally vary from one institution to another. Different institutions may have a different emphasis regarding the soft constraints which are used to evaluate an exam timetable solution. Even within the same university, exam timetabling problems may vary between faculties as the number of students and exams may differ. In addition, exam timetabling problems in universities are generally becoming more difficult due to the growing number of students. Therefore, in this thesis, a basic exam timetabling problem model is constructed, and the initial construction result is optimized to meet more soft constraint requirements.

The hyper-heuristic is described as using a metaheuristic to operate on a set of metaheuristics to solve target problems. Burke et al. (2003) argue that the Hyper-heuristics is intelligent during the process of selecting the suitable heuristic or algorithm for a given problem. The hyper-heuristic consists of two levels of heuristics and an evaluation function. To apply an existing hyper-heuristic to a new problem, only the low-level heuristics and evaluation function must be modified while the framework remains unchanged.

The goal of this thesis is to provide a more general approach/system that can be used to solve the basic exam timetabling problems of allocating exams to time slots without causing conflicts and minimizing violations to constraints. This approach/system features learning ability to enable it to be applied to various exam timetabling problems. We proposed a Hyper-heuristic framework as a general approach to solving exam timetabling problems. To enhance the learning ability of the proposed Hyper-heuristic framework,

Machine Learning algorithms were brought to the research. Reinforcement Learning algorithms were chosen to improve the design of the heuristic selection scheme and reward functions to build the learning ability of the proposed Hyper-heuristic framework. Although there has been some research into using Reinforcement Learning in Hyper-heuristic research, previous studies have primarily used the concept of the feedback scheme in Reinforcement Learning, and their success is limited. In addition, a cooperative Hyper-heuristic is proposed to further improve the performance of the proposed Hyper-heuristic framework. The aim is to develop a general framework using learning algorithms, Hyper-heuristics and cooperative schemes which can be easily used to solve various timetabling and scheduling problems.

In this research, the implemented systems will be tested against some real-world exam timetabling benchmark examples: ITC2007. The ITC2007 benchmark is extracted from real-world universities and consists of 12 datasets with differing characteristics. Although the initial and primary intention is to provide a general approach that can be applied to various exam timetabling problems, the aim is also to prove that the proposed system is capable of solving complex, real-world exam timetabling problems and is capable of delivering competitive results at the same time. The experimental results will be compared with the best results on the ITC2007 benchmark in the literature. The aim in researching Hyper-heuristic approaches is to prove that the methods implemented can be applied across a wide range of differing timetabling problems. In a word, this thesis will emphasize the generality of the Hyper-heuristic framework for solving exam timetabling problems using Reinforcement Learning algorithms and cooperative search schemes.

1.3 Thesis Overview

This thesis consists of seven chapters. The first chapter introduces the main related area of this thesis, the motivation for the current research, and also gives readers an idea of the significance. Chapter 2 explains the basic concept of the related techniques in this thesis and provides an adequate review of the literature on these techniques. Chapter 3 introduces the target problem of this study, exam timetabling. The concept, various benchmarks and previous techniques used to solve this problem are all discussed.

Chapter 4 presents an online Hyper-heuristic approach using the Reinforcement Learning and Simulated Annealing techniques. The difference between the heuristic selection methods and reward function methods we proposed are also discussed in this chapter. In Chapter 5, a Cooperative Hyper-heuristic framework with different cooperation schemes and different agent designs will be presented. The chapter includes a comparison of the different cooperation schemes along with the details of the various exam agent designs.

Chapter 6 introduces the experiments that have been performed to study the framework proposed in Chapters 4 and 5. All the proposed heuristic selection methods and reward function will be tested and compared to facilitate implementation of the best RL-SA-HH. In addition, the two different cooperation schemes described in Chapter 5 will be compared, and the best result of the proposed RL-SA-HH and COHH will be compared with the best results of other approaches in the literature. Chapter 7 summarizes the work of this thesis and draws conclusions based on the results achieved so far. Areas for future research are also suggested in this chapter.

In the next chapter, metaheuristic and Reinforcement Learning will be introduced in detail with reference to the relevant literature. The literature on the previous approaches used to solve exam timetabling problems will also be reviewed.

CHAPTER 2. BACKGROUND RESEARCH

2.1 Introduction

This chapter briefly introduces two metaheuristics that have been used in the proposed Hyper-heuristic approaches in this thesis: Simulated Annealing and Squeaky Wheel Optimization algorithms. A detailed introduction and review of the literature relating to the Hyper-heuristic and Reinforcement Learning algorithm are also presented. As for the exam timetabling problems, a detailed literature review of previous research approaches is provided.

This chapter mainly consists of four sections: introduction to the metaheuristic, discussion of the Hyper-heuristic, definition of the Reinforcement Learning approach and a review of the literature on exam timetabling problems.

2.2 Metaheuristics

A metaheuristic algorithm is a problem-independent algorithmic framework which guides the development of optimization heuristic algorithms (Sörensen, Glover 2013). Popular algorithms, including Tabu search (Xu, Chiu et al. 1998), ant colony (Dorigo, Birattari 2011) and genetic algorithm (Whitley, 1994) as well as Simulated Annealing (Ingber, 1993) are all metaheuristics. The common characteristics of these algorithms are based on mathematics and have a strong constraint on solving problems. Although the traditional optimization algorithm (refer to exact algorithm, details can be find in section 1.1) can

obtain the optimal solution for each instance of the problem, the price paid for this is unpredictable. Time overhead and result quality are often incompatible in practical application problems, so researchers began to explore and use heuristic algorithms.

Different researchers have different views on the definition of heuristic algorithms. For example, from the algorithm generated under the empirical methods, the solution can be draw with control of consumption (how much time and space). Empirical structure refers to combines algorithm development and experimentation during the algorithm design process. It is believed the difference between the expected solution and the actual result can be easily predicted in (Reeves, 1995). The heuristic algorithm is a kind of technology which is designed to reduce the human intervention and the cost of finding the most suitable solution. However, the most suitable solution is not necessarily feasible or optimal. Often, there may not even be similarity between this solution and the optimal solution (Widrow, Gupta et al. 1973). Opinion is split on whether the solution obtained by the heuristic algorithm is optimal, but this does not mean that the two viewpoints are contrary to each other, but rather that they are based on different perspectives on how to summarize the Heuristic algorithm. However, for a long period, heuristic algorithms have not been taken seriously by experts (Keliy 1996).

With the development of optimization algorithms, by 1970 the heuristic algorithm had begun to attract more attention and was regarded as an alternative method for trying to solve optimization problems (Van Breedam 2001). However, the heuristic algorithm can be designed based on experience rather than theory and can't generally achieve the optimal solution to different problems. Moreover, it has its limitations and can easily end up with the local optimum. Therefore, the optimized heuristic can be used to settle a dispute. The

heuristic algorithm is based on a thorough understanding of the essence of the problem, that is, the problem description is more restrictive. Given that some problems are difficult to solve using heuristic algorithms, researchers have put forward various metaheuristic algorithms for solving combinatorial optimization problems.

The metaheuristic algorithm (Keliy 1996), also refer to as the intelligent optimization method, is a general-purpose heuristic algorithm. It has had significant success in solving complex combinatorial optimization problems, especially for NP-hard problems. The metaheuristic algorithm is another demonstration in the natural world. It is a different method and is closely related to knowledge of biology, physics, mathematics and artificial intelligence. This algorithm can avoid the issues that arise when using a traditional heuristic algorithm like the local optimal phenomenon or the problem of generating a limited number of solutions. It is used to guide the traditional heuristic algorithm related to the problem to optimize the search space and search strategy, so as to obtain a better solution. It can be applied to many examples and has the advantage of robustness.

With the continuous development of artificial intelligence-related technologies, metaheuristic algorithms are constantly improving and optimizing. Battiti and Mascia (2007) identified the differences between traditional optimization methods and metaheuristic algorithms as follows:

1. The metaheuristic algorithm is more relaxed in the constraints of the optimization problem;
2. Traditional optimization methods are oriented, and they pay more attention to the optimality of the theory, whereas the metaheuristic algorithm focuses more on the

efficiency of the calculation.

3. Traditional optimization methods (exact methods) cannot guarantee that the algorithm will come up with a solution for every problem. However, the metaheuristic algorithm can achieve this;
4. Traditional optimization methods are based on precise mathematics, so accurate data is strictly required, but the metaheuristic algorithm is more relaxed about the quality of the data.

In light of these differences, the metaheuristic algorithm is the best choice for solving complex combinatorial optimization problems. To sum up, the metaheuristic search algorithm has the following outstanding advantages (Keller, Poli 2007):

1. This algorithm is a type of approximate algorithm, most of which can be classified as non-deterministic algorithms;
2. It can search for neighbourhoods based on heuristic search strategy;
3. It can make up for the inability of the search process to end up with the local;
4. It covers both local search and heuristic processes;
5. Its constraint on problem description is relatively small;
6. Its purpose is to effectively search for the neighbourhood and obtain the optimal solution or approximate optimal solution.

The optimization process of the metaheuristic algorithm is not dependent on the required solution, so it is widely used in existing combinatorial optimization problems. At present, research on metaheuristic algorithms in the industry is in a period of rapid

development. Related algorithms have also been developed and applied in the military field global economics and other fields. The use of more metaheuristic algorithms includes Simulated Annealing (Ingber, 1993), Tabu search (Xu, Chiu et al. 1998), iterative local search (Lourenço, Martin et al. 2003) and so on.

2.2.1 Classification of Metaheuristics

Metaheuristics arise from the real world in different ways. Some of them are based on the optimization process using a metaphor, including animal behaviours (swarm optimization), the crystalline solid cooling process (Simulated Annealing), human events (hill climbing) and even natural evolution (evolutionary algorithm) (Sörensen, Glover, 2013). The names of other metaheuristics are based on a straightforward searching structure such as Tabu search (Glover 1989).

In (Sörensen, Glover 2013), the metaheuristic approaches are divided into four different categories based on the solution manipulation strategies. These categories are:

- **Local search metaheuristics** involve searching for a better solution by optimizing a single complete solution iteratively. Examples include Iterated Local Search (ILS) (Lourenço, Martin et al. 2003), Variable Neighbourhood Search (VNS) (Hansen, Mladenovic et al. 2003), Tabu list search (Glover 1989) and Simulated Annealing (SA) (Nahar, Sahni et al. 1986).
- **Constructive metaheuristics** involve searching for solutions through construction instead of optimization. Examples include Ant Colony Optimization (ACO) (Dorigo, Birattari 2011), Squeaky Wheel Optimization (SWO) (Joslin, Clements 1999) and GRASP (Marques-Silva, Sakallah 1999).

- **Population-based metaheuristics** involve searching for a better solution through selecting and combining an existing set of solutions iteratively. Examples include Genetic Algorithm (GA) (Whitley 1994) and Evolutionary Algorithm (EA) (Back 1996).
- **Hybrid metaheuristics** is a metaheuristic framework which includes a construction phase as well as an optimization phase. Examples include Memetic Algorithm (MA) (Moscato, Cotta et al. 2004) and GRASP (Feo, Resende, 1995).

In the research for this thesis, two metaheuristics are used to help build the proposed Hyper-heuristic framework: Squeaky Wheel Optimization and Simulated Annealing algorithm. In the following section, a detailed introduction and literature review are provided for these two heuristics.

2.2.2 Squeaky Wheel Optimization

Proposed by Joslin and Clements (1999), Squeaky Wheel Optimization takes its name from proverb, "the squeaky wheel gets the grease", meaning that the most noticeable problems tend to attract the most attention. For a given problem, the SWO algorithm builds a solution through a greedy algorithm. By analysing the intrinsic properties of the problem, it analyses each element of the solution, distinguishes the difficulty of processing, finds the point of failure and then specifically assigns a value to the corresponding penalty. When the sorter sorts the results of the analysis, the location of these intractable elements will be closer to the front so that when a new solution is created, this difficult element will be prioritized. The purpose of this analysis is to generate a new priority order and then enable the

constructor to construct the next solution. This "build/analyse/sort" loop is repeated until the limit is reached or an acceptable solution is found. SWO can be regarded as the process running in the search or the sort space of the solution space. The characteristics of this "coupled spatial search" are very significant'. Since the new solution is indirectly obtained by analysing the previous solution and reordering it, the construction and analysis are key to the next solution. Constructors, Analysers and Sorters are the key components of the SWO algorithm.

In many of the metaheuristics reported in recent years, SWO (Joslin and Clements, 1999) has been relatively unknown but it is receiving increasing attention in the research field. This approach is based on the following observations. In real-world combinatorial problems, solutions are composed of components that are intricately woven in a nonlinear, non-addictive manner. In order to deal with these components, Joslin and Clements (1999) originally proposed and applied SWO to production line scheduling problems as well as graph colouring problems, with some satisfactory results. Since then, SWO has been successfully applied to a number of different types of discrete optimization problems, such as airspace communication (Barbulescu, Watson et al. 2004), educational scheduling (Burke er al., 2004), personnel scheduling (Aickelin, Burke et al. 2009), port space distribution (Fu, Li et al. 2007), multi-tone bandwidth (Malaguti, Toth 2008), machine scheduling (Feng, Lau 2008) and strip packing (Burke, Edmund K., Hyde et al. 2011).

In SWO, the initial solution is first constructed by a greedy algorithm. The solution is then analysed to assign the error to the component that causes the solution "trouble", and this information is used to modify the priority order of the greedy algorithm to construct the new solution. Structural analysis continues in priority order until the stop condition is

reached. Consequently, the result of the SWO cycle is difficult to deal with. The problem components tend to rise in a priority queue, and the components that are easy to handle tend to sink. In essence, this method finds high-quality solutions by searching two spaces at the same time: traditional solution space and new priority space.

However, as a constructive metaheuristic algorithm, SWO has its own shortcomings, such as poor scalability due to the random starting point of the greedy constructor, and slow convergence due to the inability to make small movements in the solution space. If the construction process starts from only part of the solution to each cycle, SWO will accelerate significantly. In addition, if the change of components can be limited to the fault maker, the corresponding solution will be changed relatively little. To solve these problems, Aickelin and Li (2007) proposed an evolutionary version of SWO and reported two human scheduling problems of driver scheduling (Li, Kwan 2003) and nurse scheduling (Aickelin, Li 2007) which were significantly better than the original SWO. To achieve this goal, evolutionary SWO (ESWO) meets two additional operations in cycles: selection and mutation. The selection operator determines whether to abandon components and place them in the newly allocated queue according to their local fitness. Mutation steps further discard some randomly selected components.

The structural framework of the SWO algorithm is shown in Figure 1. The core of the SWO algorithm is mainly composed of three components:

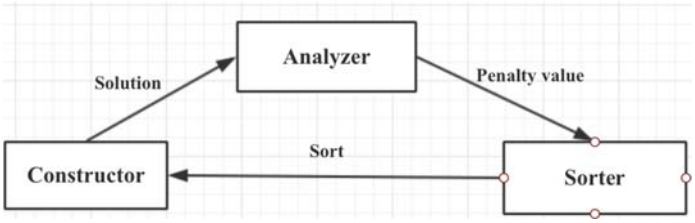

Figure 1:Framework of SWO (Joslin and Clements, 1999)

- **Constructor**. For problem elements that are given a priority sequence, the constructor draws up solutions that use a greedy algorithm without backtracking. The sequence determines the order or solution.
- **Analyser**. The Analyser will specify the penalty value for the current problem element. This penalty value helps to resolve the defects of the current solution. For example, if the optimization goal in the exam timetabling problem requires another late arrival time, then for latecomers, the Analyser will assign the corresponding penalty value according to its lateness time. A key principle behind the SWO algorithm is to reveal the problem structure of the current solution. By analysing the solution, it is possible to determine which elements in the current plan are good elements and which elements are bad. The information provided by the problem structure is local, because it may only be applicable to a part of the space currently being searched, but it is useful for the sequencer to determine the direction of the next search.
- **Sequencer**. The sorter sorts based on the results obtained by the Analyser, that is, the constituent elements in the problem are assigned a priority sequence based on the culprit assigned by the Analyser. Based on the principle of giving priority treatment to difficult elements, the more difficult the elements are to deal with, the

greater the value of the penalty they are assigned to, which advances their position in the priority sequence. Thus, when forming a new solution in the constructor, it will prioritise dealing with these large elements of punishment. After the difficult element is resolved, its penalty value will decrease, that is, its position in the priority sequence will fall, and the current difficult element will advance in the priority sequence due to the higher penalty value. The sorter is actually the direction in which the constructor handles the problem element.

Finding an optimal solution is difficult in the actual process of the problem. It is possible to use static analysis to identify the difficulty of the problem, but because the interaction between the different elements can be complex, the elements that are causing trouble currently may be easily handled at another time. SWO is not trying to find difficult elements in the global scope, but to search for difficult elements through a subdivided solution in the local space. Through this analysis, difficult elements in the global scope will be determined over time, because it is difficult for these difficult elements to escape a search of the local space.

As shown in Figure 2, SWO search occurs in two coupling spaces: one is a familiar solution space, and the other is a priority space. The movement within the solution space is indirect and is mainly achieved by analysing the results of existing solutions and re-prioritizing them.

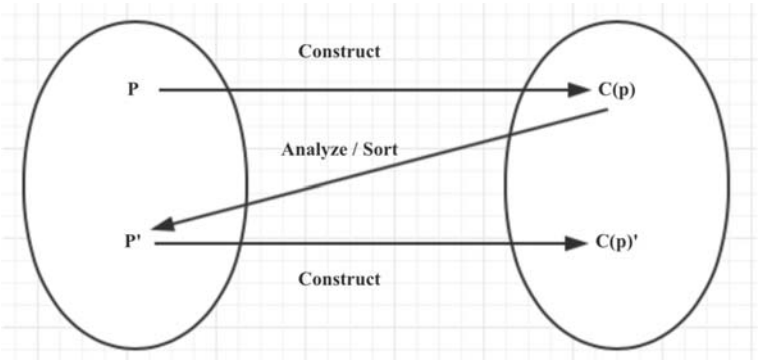

Figure 2:Coupling Spaces in SWO (Joslin, Clements 1999)

SWO can quickly find a better solution by searching in two spaces at the same time. The combination of the old and new spaces prevents the problems that readily appear in many local search algorithms, enabling SWO to make coherent and large movements within the search space and jump out of undesired areas, thereby avoiding local optima. The "analyzer/sequencer/constructor" loop causes the difficult part of the problem of rising in the priority sequence, while the easy-to-handle part falls into the priority sequence, eventually forming a solution.

2.2.3 Simulated Annealing

Simulated Annealing was put forward by Metropolis et al. (1953). Now it has been widely applied in engineering. The main idea is inspired by the simulation of the solid annealing cooling process, which warms up the solids to a full height and allows them to cool slowly. When the solids are heated, the thermal motion of the atoms in the solids increases and the internal energy also increases. As the temperature rises, the long-range ordering of the solids is completely destroyed, and the solid internal particles become disorderly as the temperature increases. Upon cooling, the particles anneal gradually in order to be in an

equilibrium, and finally reach the ground state at room temperature. In the meantime, the internal energy is minimized.

In practical applications, we can simulate the internal energy E as the objective function value f, simulate the temperature T as a control parameter, and then start from a given solution and randomly generate a new solution from its neighbourhood. The acceptance criterion allows the solution to worsen the objective function within a certain range, and the algorithm continuously performs an iterative process of generating a new solution—i.e., calculating the difference using objective function and judging whether to accept the new solution—and accepting or rejecting it, corresponding to the process whereby a solid with a constant temperature tends towards thermal balance. After a large number of solution changes, it is possible to obtain a relatively optimal solution to the optimization problem when given the control parameter T, then reduce the value of the number T, the control parameters, and repeat the iterative process described above. As the control parameters gradually decrease to zero, the system tends to become more and more balanced. Eventually, the system state corresponds to the overall optimal solution of the optimization problem. The annealing process is controlled by the parameter in a cooling schedule, including an initial value T and the attenuation factor *delta t*, the number of iterations for each value of T, L and the termination condition S.

The Simulated Annealing algorithm (SA) has the following characteristics compared to other search methods:

Characteristic 1: Receives a worse solution with a certain probability

The SA is different from the traditional random search method in the search strategy. It not only introduces appropriate random factors, but also introduces the natural mechanism of the annealing process of the physical system. The introduction of this natural mechanism makes the SA not only accept the good tentative point that improves the objective function in the iterative process, but also accept the tentative point that causes the objective function value to be "poor" with a certain probability. The state is randomly generated, does not force the latter state to be better than the previous state, and accepts that probability gradually increases with the decrease in temperature. The traditional method is often the iterative search process of the optimal solution starting from an initial point in the solution space. For example, when hill-climbing, if a slight change can improve the quality of the final searched solution, then go in that direction, otherwise, take the opposite direction. However, complex problems will generate several local optimal solutions. Traditional methods are easily confined to locally optimal solutions and stagnate. Many traditional optimization algorithms are often deterministic. The transfer from one search point to another has a definite transfer method and transfer relationship. This determinism may prevent the search from ever reaching the best point, thus limiting the application of the algorithm. The SA searches in a probabilistic way, increasing the flexibility of the search process using an acceptance probability to control the search. .

Characteristic 2: Introduces algorithm control parameters

An algorithm control parameter similar to the annealing temperature is introduced, which divides the optimization process into various stages and decides the criteria for selecting

random states at each stage. The acceptance function is given in the algorithm proposed by Metropolis et al. (1953) as a simple mathematical model. The two important steps of the SA are as follows:

1. First, starting from the previous iteration point under each control parameter, a neighbour random state is generated. The acceptance criteria determined by the control parameters determine the trade-off of the new status, and thus from the random length of the Markov chain;

2. Second, the number of control parameters is slowly reduced, the receiving criteria are improved until the control parameter tends to zero and the state chain is stable in the optimal state of the optimization problem, thereby improving the reliability of the SA global optimal solution.

Characteristic 3: Search using object function values

The traditional search algorithm not only needs to use the value of the objective function, but also needs some auxiliary information to determine the search direction, such as the derivative value of the objective function. When no such information exists, the algorithm fails. The SA uses only the fitness function value transformed by the objective function to determine the further search direction and search range with no additional auxiliary information. It is important to point out that the fitness function of the SA is not limited by continuous differentiability, and its domain can be set arbitrarily. The only requirement for the fitness function is that the input can be calculated to the comparing output. This feature is more convenient for the application of the SA for many functions that cannot be derived

or are difficult to derive, or for functions that do not have derivatives, as well as combinatorial optimization problems. In addition, by directly using the value of the objective function or individual fitness, the search range can be concentrated on the part of the search space with higher fitness. This will increase search efficiency.

Characteristic 4: Implied parallelism

The parallel algorithm was developed in the 1960s and the speed of its development has been very fast. Some experts even believe that the only way to improve the performance of computer systems at present is to choose a great deal of parallelism. Currently, the design of the parallel algorithm mainly adopts two methods: one is the transformation of the existing serial algorithm to make it a good parallel algorithm; the other is to directly design the new parallel algorithm by combining the structural features of the parallel computer used. It is still relatively easy to transform the SA into a parallel algorithm. At present, there are several parallel strategies: parallel operations strategy, parallel audition strategy, regional split strategy, and loose relaxation strategy. These parallel algorithms are superior to the SA in the quality and convergence rate of the solution to different extents. Therefore, it can be foreseen that the large-scale parallel computing model will become the mainstream of the study of global optimization problems.

Characteristic 5: Search for complex areas

The SA is best at searching complex areas and finding areas with high expected values, but it is inefficient in solving simple problems.

2.2.3.1 Cooling schedules

The so-called cooling schedule is a set of parameters that control the process of the algorithm to approximate the asymptotically convergent state of the SA so that the algorithm returns an approximately optimal solution after a time-limited execution process. The cooling schedule includes the initial value of the control parameter, its decay function, the length of the corresponding Markov chain and the stopping criteria. It is an important factor influencing the experimental performance of the SA, and its reasonable selection is the key to the application of the algorithm.

Initial value of control parameters: t_0

Judging from experience, the algorithm process should search for the largest possible solution space within a reasonable time period. Only a sufficiently large initial value can meet this requirement, and the initial acceptance rate should be approximately equal to 1. According to Metropolis guidelines:

$$\boldsymbol{exp}(-\frac{\Delta \boldsymbol{f}}{\boldsymbol{t_0}}) \approx \mathbf{1}$$

we can infer that the initial value of control parameters t_0 should be great.

Markov chain length

The Markov chain is an attempted sequence in which the outcome of an attempt depends only on the outcome of the previous attempt and thus has a memory forgetting function. The length of the Markov chain represents the number of transformations produced by the

k-th iteration of the Metropolis algorithm. The principle of Markov chain length selection is as follows: assuming that the attenuation function of the control parameter t has been selected, the selection of the length of the Markov chain should satisfy the requirement that the probability distribution of the solution tends to be stable at each value of the control parameter.

Distributed digital conversion stabilized at a value of each control parameter. This parameter may need to be calculated by converting the number which should be at least acceptable recovery of the stationary profile (some fixed number).

However, since the acceptance rate of the transformation decreases with decreasing control parameter values, the number of transformations that are required to receive a fixed number of transformations increases. Finally, when $t_0 \to 0$, $L_k \to \infty$. Thus, we can use some constant $\bar{L}$ to limit the value of L_k and thereby avoid producing a long Markov chain in the small value of t_k.

L_k gives the number of transformations in the k-th Markov chain, the finite sequence $\{L_k\}$ specifies the scope of the solution process for the algorithm process search. In the next section, the method to determine the value of L_k is discussed.

Fixed length

As the spatial solution scale $|S|$ of most combinatorial optimization problems increases with the scale of the problem n exponentially, in order to ensure the quality of the final solution of the SA, the connection between L_k and n can be established. The exponential relationship is clearly impractical, so $\bar{L}$ is usually taken as a polynomial function of problem size n. Hajiaghayi and Kirkpatrick et al. (2014) adopted the directly specified $\bar{L} =$

n. Using a fixed length L is a simple method. The value of L_k is irrelevant with the control parameter. For a given combinatorial optimization problem, it is a constant that does not change with the algorithm process.

The ratio of acceptance and rejection

When the control parameter value is large, the frequency of acceptance of each state is basically the same, and almost all the states are accepted. At this time, the number of iteration steps can be made as small as possible under this control parameter value. As the control parameter value gradually becomes smaller, more and more states are rejected. If the number of iterations under this control parameter value is too small, it may result in a premature falling into a locally optimal solution. The more intuitive and effective method is to increase the value of L_k when decreasing the value of the control parameter. One way to achieve this is to give an acceptance index i. When the acceptance number is equal to i, the Markov chain under the control parameter value ends, and the control parameter value decreases.

The second method is to give an acceptance rate q and record the total number of iterations and the number of times that new moves are accepted under this control parameter value. When the acceptance rate is not less than q, the Markov chain under the control parameter value ends, and the control parameter value decreases. Using this method, the value of L_k is related to the value of the control parameter. A small attenuation is generally chosen to avoid excessively long Markov chains. This type of method makes it easier to control the quality of the final solution, but it is difficult to grasp the CPU time

of the algorithm process. If an inappropriate L_k is chosen, it may even jeopardize the convergence of algorithm.

Attenuation function of control factors

To prevent the algorithm process from generating a long Markov chain, the degree of attenuation of the control parameters must be small. It is believed that in the case of small attenuation of the control parameter, the smooth distribution of two consecutive values t_k and t_{k+1} is close to each other. So, if the value t_k has reached a steady distribution, it can be expected that after t_k attenuates to t_{k+1}, it may be sufficient to recover the steady distribution of value $t_{k+1}$with only a small degree of transformation. This allows shorter length Markov chains to be used to reduce CPU time.

The small degree of attenuation of the control parameter may also lead to an increase in the number of iterations of the algorithm process so that the algorithm process can be expected to accept more transformations, access more neighbours, search a larger range of solution space and return a higher-quality final solution. It also takes more CPU time. Experiments show that as long as the attenuation function is selected properly, the quality of the final solution can be greatly improved without affecting the reasonableness of the CPU time. The commonly used control parameter attenuation functions are as follows:

(1) $t_{k+1} = \alpha t_k, k = 0, 1, 2 \ldots \ldots$

where α is a constant that is close to 1. This attenuation function was first proposed in (Abramson, Krishnamoorthy et al. 1999). This method is simple and easy to use and is popular among users. We usually take$\alpha = 0.5\ to\ 0.99$. The attenuation of the control

parameter by the attenuation function is decremented with the algorithm process.

(2) $t_k = \frac{L-k}{L}, k = 1,\ 2 \ldots\ldots L.$

where L is the total number of times the algorithm controls the parameter drop. The attenuation function can simply control the total number of control parameter drops, so that the difference between the successively controlled values of the control parameters remains unchanged. This method is only suitable for the cooling schedule with the criterion that the stop is based on the number of iterations L.

Of the two types of attenuation functions above, the first has a decrement that decreases with the algorithmic process, so that the rate of decrement of the control parameter value can be reduced, thereby delaying the declining trend of the transform acceptance probability of the algorithm process, which is undoubtedly beneficial to the test performance stability of the SA. In the second type, by contrast, the temperature decrement is mainly affected by the number of iterations.

Termination criteria

A reasonable termination criterion is to ensure that the algorithm converges to an approximate solution and that the final solution has a certain quality. Considering the limited CPU time, Nahar et al. (1986) proposed to use the number of pre-determined control parameters, or to be specific, the number of Markov chains or the iteration number *k* as a termination criterion. Control over the quality of the final solution under termination criteria constructed using iterations is weak and lacks flexibility. Controlling the asymptotic convergence of SA provides new insights: The convergence of the algorithm to the optimal solution set is gradual as the value of the control factor *t* decreases slowly.

Only when t is "small enough" can a high-quality optimal solution be possible. Therefore, a "small enough" t can replace the criterion of "final solution quality" to some extent, which can be used as a termination criteria.

1. One is to make the control parameter t value less than a certain positive number e, which directly constitutes the stopping criterion of the formula $t < e$.
2. The second is to determine a termination parameter x with the behaviour of the algorithm process. Acceptance probability decreases with the control parameter value; if the current acceptance rate of the algorithm process p_k<x, the algorithm is terminated. This method takes into account both the final solution quality and the CPU time requirement for stopping criteria. As long as the value is properly selected, the CPU time can be reduced, and the quality of the final solution is still guaranteed.
3. Another common way to choose termination criteria is not to improve the rule control method and instead to use some approximate solutions obtained by the algorithm process as a measure to determine whether the quality of the current solution of the algorithm is continuously and significantly improved, so as to determine whether to terminate the algorithm. For example, the algorithm can be terminated in several successive Markov chains when the solution is unchanged. This method also takes into account the quality of the final solution and CPU time. Furthermore, with the cooperation of t_k and L_k, not only can a high-quality final solution be expected, but also the relative control of CPU time (i.e., the CPU time increases with the size of the problem), and the solution quality is relatively stable.

2.2.3.2 Reheating scheme

The Reheating Scheme is part of the Simulated Annealing schedule. Abramson et al. (1999) proposed two different kinds of reheating scheme: geometric reheating and enhanced geometric reheating.

The geometric reheating process aims to solve the contradictions in the cooling schedule: when the temperature is at a low level, it cannot meet the requirement of Simulated Annealing to increase the temperature to seek the global optimum. Thus, this scheme works when it automatically detects the local minima and avoids it. This scheme can be shown as:

$$Cooling: T_{k+1} = \alpha T_k \text{ where } \alpha \textit{ is constant and less than } 1$$

$$Heating: T_{k+1} = \frac{T_k}{\beta} \text{ where } \beta \textit{ is constant and less than } 1$$

The core of this scheme is to determine when to swap between heating and cooling. Generally, the cooling schedule is terminated when the system cannot detect an improvement in cost, and the reheating scheme is terminated when the improvement in cost increases or decreases. However, this scheme also has its defects. It may be caught in local optima cycles.

Enhanced geometric reheating, while similar to geometric reheating, strengthens that scheme to avoid excessive heating and avoid getting caught in cycles. It chooses a good value of heating factor that is closer to 1 when the local optimum occurs. If the cycle still exists after Markov chains, the value of the heating factor will be reduced. This scheme can be shown as:

$$\beta = \beta\_default \text{ where } \beta\_default \ is\ a\ large\ constant\ and\ less\ than\ 1$$

$$Cooling: T_{k+1} = \alpha T_k \text{ where } \alpha\ is\ constant\ and\ less\ than\ 1$$

$$Heating: T_{k+1} = \frac{T_k}{\beta} \text{ where } \beta\ is\ constant\ and\ less\ than\ 1$$

$$\beta = \beta - \varepsilon\ when\ Count(local_{optima}) > Fixed\_number$$

The termination equation can be shown as follows:

$$\text{if } \Delta C_k > 0\ and\ \Delta C_{k+1} = -\Delta C_k\ or\ \Delta C_{k+1} = -2\Delta C_k\ then\ fluctuation = true$$

$$where\ \Delta C_k = C_k - C_{k-1}, \quad \Delta C_{k+1} = C_{k+1} - C_k,$$

$$C_k\ is\ the\ cost\ at\ the\ end\ of\ the\ k'th\ Markov\ chain$$

2.3 Hyper-Heuristic

At present, the main methods for solving combinatorial optimization problems include precise algorithms and heuristic algorithms. The exact algorithm can achieve the optimal solution to the problem, but the calculation time increases exponentially with the scale of the problem. The metaheuristic algorithm has been widely used to solve all kinds of combinatorial optimization problems, but there are some issues associated with this method.

First, the metaheuristic algorithm is often designed to solve a specific problem and lacks generality. Secondly, the ability of the algorithm designer needs to be high and include professional background knowledge and algorithm design skills. Finally, the actual applications are very different. According to the “no free lunch theory” (Wolpert,

Macready 1997), an algorithm which is good in some cases is often not good in others. In order to solve the above problems, the concept of the Hyper-Heuristic Algorithm was proposed and quickly attracted the attention of the academic community. Some scholars have applied it to problems of combinatorial optimization, such as scheduling problems and packing problems.

Cowling and Kendall et al. (2001) first proposed the Hyper-heuristic algorithm and used it to solve the various scheduling problem (Cowling, Kendall, and Soubeiga 2002; Cowling, Kendall, and Han 2002). The authors described the Hyper-heuristic algorithm as "a heuristic algorithm for finding heuristic algorithms". Burke, Edmund K. and Hyde et al. (2010) define the Hyper-heuristic algorithm more precisely: "The Hyper-heuristic algorithm provides a high-level heuristic method for solving various combinatorial optimization problems by managing or manipulating a series of low-level heuristics."

A typical Hyper-heuristic algorithm consists of two parts: the control domain and the problem domain in the logical structure, as shown in Figure 3. The problem domain contains the problem described by the experts in the field of design its basic functions and evaluation function, as well as several low-level heuristics. A high-level strategy is designed by the experts of the Hyper-heuristic algorithm. The algorithm expert is designed to solve problems, including how to use low-level heuristics to construct feasible solutions and how to improve the quality of those solutions. There is a domain mask between the problem domain and the control domain, which defines a standard interface for information transfer between two layers of the structure.

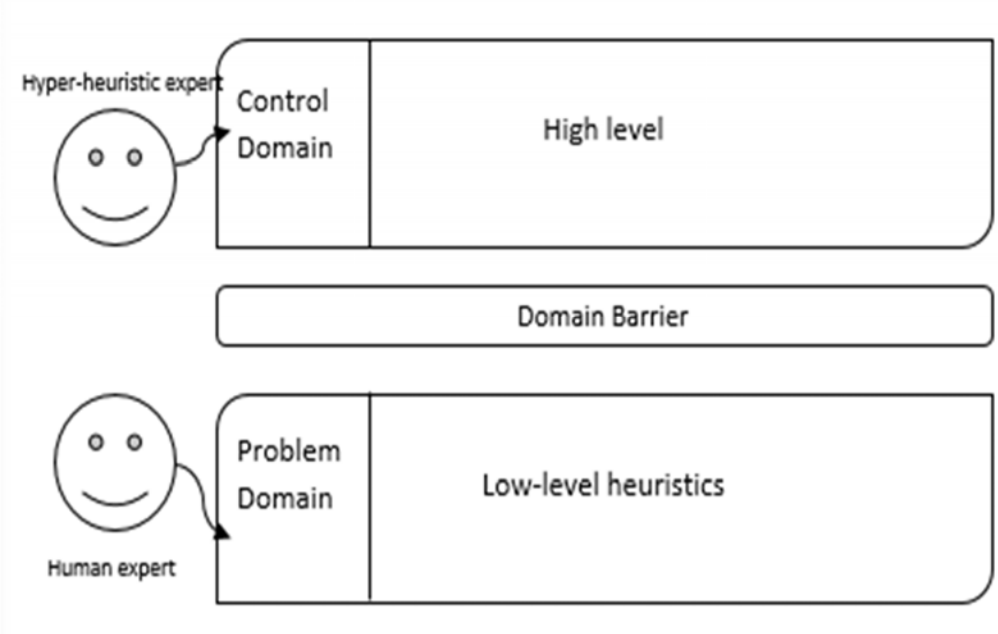

Figure 3:Structure of Hyper-heuristic Algorithm (Burke et al., 2010)

2.3.1 Hyper-Heuristic Framework

A Hyper-heuristic Flexible framework (HyFlex) is proposed in (Burke, Edmund K., Curtois et al. 2009), which uses Java language to provide a common interface to solve combinatorial optimization problems in different problem areas. In this framework, programmers need not be concerned about the problem domain itself, such as the underlying data structure and low-level heuristics. Instead, they can focus on designing general high-level strategies. Currently, the framework offers several types of question descriptions: Bin packing, Flow Shop, SAT, CVRP, Personnel Schedule and TPS, and provides corresponding low-level heuristics, data structures and benchmark cases for each problem area. Researchers only need to design, select or mix these low-level heuristic algorithms.

Burke et al. (2009) use HyFlex to organize the Cross- Domain Heuristic Search Challenge (CHeSC) competition, in which contestants design high-level strategies for Hyper-heuristic algorithms, and then evaluate the performance of high-level strategies in relation to various aspects, such as time efficiency and optimization capabilities. At present, some scholars use the HyFlex framework to study the Hyper-heuristic algorithm framework. Burke, and Edmund K., Gendreau et al. (2011) designed a high-level control strategy based on online learning. Özcan and Kheiri (2011) proposed a Hyper-heuristic based on a random gradient. Although the strategy is relatively simple, good experimental results were obtained. Widrow and Gupta et al. (1973) combined the use of Reinforcement Learning and adaptive threshold acceptance methods; they designed an adaptive threshold acceptance method and achieved good computer results combined with low-level heuristic algorithm parameters adaptive mechanism. Hsiao and Chiang et al. (2012) adopted an adaptive variable neighbourhood search framework. The authors divide the neighbourhood search process into two phases of diversity and concentration and adjusted the scale of the neighbourhood using adaptive techniques. Asta and Özcan (2014) proposed an offline learning algorithm to solve the vehicle routing problem in HyFlex. The algorithm is divided into two phases. In the first phase, some experts training algorithm specifies instance within a limited time, and records the depth of the search algorithm, the parameter values and the variation of the conditions during the receiving solution. In the second stage, the trained results solve for the unknown cases. Kheiri and Özcan (2016) used a multi-stage iterative search algorithm and a selection function method.

HyFlex provides a new way for researchers to use Hyper-heuristic algorithms. On the one hand, it provides an open source framework that contains six problem domains.

Researchers can focus on the design and implementation of high-level control strategies. On the other hand, it uses benchmark cases for testing, not only providing the opportunity for lateral comparison of strategies of various high-level controls, but also verifying that the performance of the Hyper-heuristic algorithm is competitive with traditional heuristic algorithms. It shows that the Hyper-heuristic algorithm has the advantages of universality and the ability to obtain high-quality solutions.

In Soubeiga (2003), the Hyper-heuristic algorithms are divided into two categories: with learning and without learning. The Hyper-heuristic without learning mechanism calls the low-level heuristic operator based on pre-determined sequences. A typical example of this type is the multivariable domain search (Mladenović, Hansen 1997). Learning-based Hyper-heuristic algorithms can dynamically change their preference based on the historical performance of each low-level heuristic operator. As Soubeiga (2003) demonstrates, Hyper-heuristic algorithms can be further classified based on different learning strategies. In the literature, Hyper-heuristic methods are divided into constructive Hyper-heuristic methods and local search Hyper-heuristic methods. The constructive Hyper-heuristic method incrementally constructs a solution by selecting appropriate low-level heuristic operators. These low-level heuristic operators all belong to construction class operators. The so-called construction class operator is a kind of operator that can create solutions from scratch and can construct a complete solution step by step. The corresponding local search Hyper-heuristic method starts from a complete initial solution, and iteratively selects the appropriate local search, low-level heuristic operator, hoping that the search results are optimized.

In Chakhlevitch and Cowling (2008), the Hyper-heuristic method is divided into four types: (1) a Hyper-heuristic method based on a random selection of low-level heuristic operators; (2) a greedy Hyper-heuristic method, which involves finding all the evaluation values that low-level heuristic operators act on the current solution, then choosing the one that performs best; (3) a Hyper-heuristic method based on a metaheuristic method; and (4) a Hyper-heuristic method that uses a learning strategy to manage low-level heuristic operators. In the new classification method, the Hyper-heuristic method is classified according to two dimensions: the attributes of the heuristic search space and the source of the feedback of the learning mechanism. The two dimensions are orthogonal, so different heuristic search spaces can be combined with different feedback sources.

The properties of the heuristic search space fall into two categories:

1. Heuristic selection: Strategies for selecting existing low-level heuristic operators.
2. Heuristic generation: This generates new heuristics from heuristic methods that already exist. Whether it is a heuristic selection strategy or a heuristic generation strategy, it can be further subdivided into a construction heuristic method and a perturbation heuristic method. Some Hyper-heuristic algorithms can be considered to be learning algorithms because they use the information fed back from the search process. However, Hyper-heuristic algorithms without learning strategies do not use the information returned during the search process.

According to the source of feedback information in the learning process, these algorithms can be classified as online learning, offline learning, and no learning:

1. Online learning Hyper-heuristic algorithms: The learning process occurs during instances in which the Hyper-heuristic algorithm is solving the problem. At this time,

task-independent attributes can be used by high-level strategies to determine which appropriate low-level heuristic operator applies.

2. Offline learning Hyper-heuristic algorithms: The core idea is to acquire knowledge from the training set, and then hope that the acquired knowledge can be applied in a general way to solve instances that have not yet been seen.

2.3.2 Agent-Based Cooperative Hyper-Heuristic

The original idea of an agent-based cooperative Hyper-heuristic was proposed by Ouelhadj and Petrovic (2010).

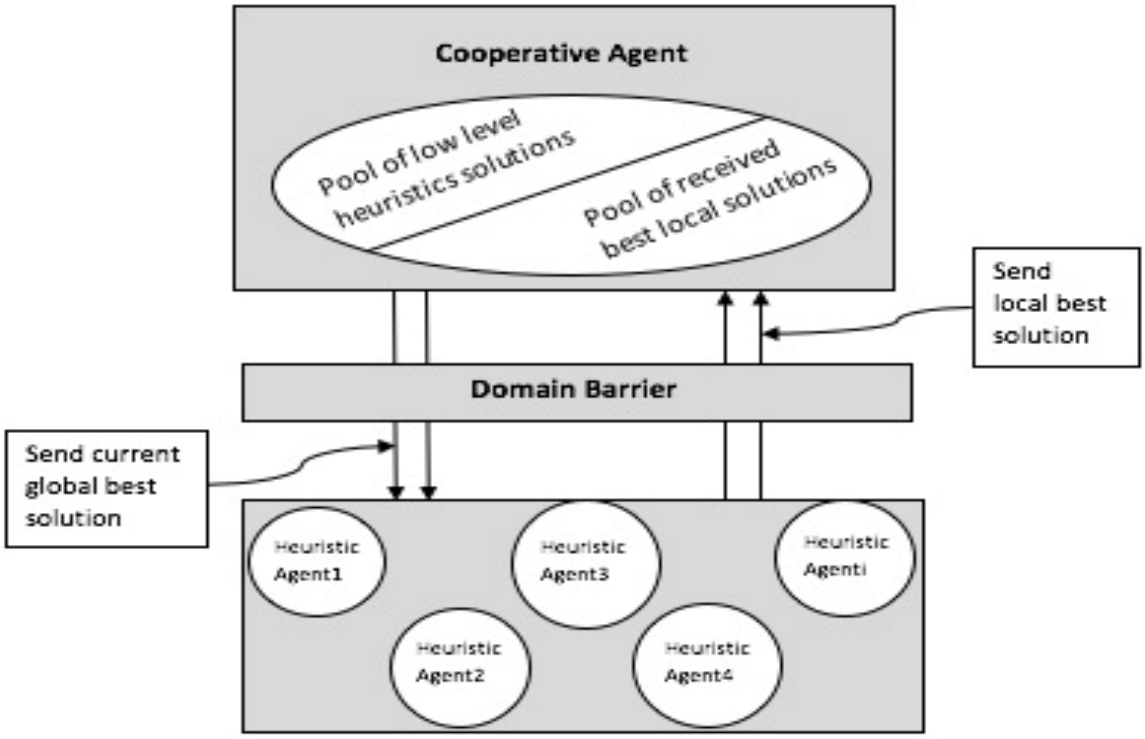

Figure 4:Framework of cooperative Hyper-heuristics (Ouelhadj, Petrovic 2010)

As Figure 4 shows, it is a system based on the agent. It consists of heuristic agents and a cooperative Hyper-heuristic agent. In the case of heuristic agents, each low agent will be distributed to the upper heuristic agents. The upper heuristic agents can begin a local search to find a complete solution so that each type of low-level heuristic method can improve its local solution. By contrast, the cooperative Hyper-heuristic agent manages the cooperation among the heuristic agents. Also, it sends the solution to the low-level heuristic agent to do further work.

2.3.3 Online Learning Hyper-Heuristic

The Hyper-heuristic method is a popular technology that has emerged in recent years. Unlike traditional methods that are used to find the right solution, it can be seen as a heuristic algorithm to find the right heuristic algorithm. As (Ross, Hart et al. 1997), for the problem of examination timetabling, suitable algorithms should be found that can solve a variety of specific problems, rather than be directly used to find the actual solution of all problems. In other words, the method provides some kind of high-level strategy. By calling the low-level heuristic operators, the final solution to the problem can be achieved.

Ahmadi and Barone et al. (2003) put forward new ideas to solve the exam timetabling problem. They found a combination to solve different problems arising from the various exam timetable heuristic methods through their variable neighbourhood search. Özcan and Alpay (2002) proposed a steady-state genetic algorithm for solving course schedule and examination schedule problems. In this algorithm, the search target of the genetic algorithm is to find a suitable heuristic algorithm. Kendall and Hussin (2004) have studied the Hyper-heuristic approach combined with the Tabu search method for solving

exam timetabling problems. This research is focused on the construction of the Hyper-heuristic module regarding how to select the best heuristic intelligently with the help of the Tabu list. Burke and Kendall et al. (2003) used a Tabu search method to find the right heuristic sequence diagram. Through this sequence, a classical algorithm to build a solution sets a timetable to solve the problem of this algorithm in the examination schedule problem, known as an ultra-biased heuristic graph. Bilgin and Özcan et al. (2006) analysed seven heuristic selection methods and five reception criteria in the Hyper-heuristic method to solve the exam scheduling problem. The authors found that using different selection methods and receiving criteria is more effective for different problems.

2.4 Reinforcement Learning

Reinforcement Learning is a branch of machine learning. It is particularly good at controlling, can act autonomously in an environment and continuously improve its behaviour through interaction with the environment, such as perception and feedback. In many other machine learning algorithms, the autonomous agents learn what to do, but Reinforcement Learning is the process of trying to learn which actions can be most rewarded in a particular situation. In many scenarios, these actions will affect not only the current return value, but also the subsequent status and a series of return values.

The three most important features of Reinforcement Learning are:

1. A closed-loop form;
2. It does not directly indicate which action to choose;
3. Series of actions and reward signals all affect the longer ones

The basic principles of Reinforcement Learning are as follows. At time *t*, the agent determines the next round of decision actions based on the current state and historical information. It then gives information on the current state and the agent's decision, and the environment determines the next round of status, the return value of the agent and other information it can observe. Finally, it iterates until the task is completed. Unlike the planning problem, the learning problem does not know the whole situation of the environment initially, that is, it is in a state with no concrete environment model. Therefore, it is necessary to gradually try the wrong learning environment and adjust its own decision-making. In relation to the mechanism of rewards, it is assumed that all goals can be characterized as the maximization of expected rewards. From the above description, it is clear that there are four key components of the Reinforcement Learning (Sutton Barto, 1998), as follows.

Component 1: Policy function

The policy function involves mapping the environmental state of behaviour. For an agent in a certain state, it needs to select an action to execute and get the reward value. To answer the question of how to choose the action application, it is necessary to form a certain strategy for selecting actions. The term for this is “policy”. It determines the optimal strategy results learned from the agent to some extent, which is the core element of the Reinforcement Learning algorithm. In the process of agent learning, the agent constantly explores the surrounding environment and obtains reward values, establishing a mapping of the environment to behaviour. The fact that the learning speed is slow during the

algorithm implementation is often caused by the unreasonable design of the exploration strategy.

Component 2: Reward function

Rewards are feedbacks that are given to an agent when it performs a behaviour. The goal of the reinforcement learning system is to maximize the cumulative rewards. Performing the same behaviour in different states may result in different rewards. The reward function determines the goal of agent learning. Only when the correct reward function is set can the agent reach the goal of learning in the course of iterative updating.

Component 3: Value function

When an agent selects an action to perform, it will receive an instantaneous reward value, which is an immediate assessment of the effect of the action. For the long-term effect of the selection action, the evaluation of the value function is more accurate. It can better reflect the relationship between the selection action and the environmental situational trait and obtain the largest reward value from the long-term perspective. For agents, the goal of learning is to maximize the accumulated reward value.

Component 4: Environmental model

The next step is to predict what changes the environment will make to predict what the agent receives or what the reward is. Thus, there are two types of models: a transition model for predicting the next episode, and a reward model for predicting the next reward value. The agent can predict the environment according to the environment model. The

Reinforcement Learning method using the environment model is called a model-based method, and the Reinforcement Learning method that does not use the environment model is known as a modeless method.

Reinforcement Learning originates from an activist conception in psychology which holds that individuals can be motivated by the punishments and rewards given by the environment, which slowly form an estimate of motivation, resulting in the biggest gains. This method has universal adaptability, and its application therefore has been explored in many other areas by experts from various countries. The Reinforcement Learning algorithm was first proposed by Samuel (1959) and was first successfully applied in chess. Subsequently, the Temporal Difference (TD) algorithm and the Q algorithm in Reinforcement Learning were proposed by Sutton (1988) as well as Watkins and Dayan et al. (1992), respectively, both of whom proposed and verified the convergence of the proposed algorithm. After the algorithm theory of Reinforcement Learning was successively put forward, research in the industry on Reinforcement Learning began to focus on practical applications. Yasunobu and Matsubara (2003) proposed that the Reinforcement Learning algorithm based on fuzzy control could be used to solve the problem of car parking control. In the robot control field, the Reinforcement Learning algorithm has obvious advantages in obstacle avoidance and footpath planning (Zamstein, Arroyo et al. 2006).

Although the industry has made breakthroughs in the research on Reinforcement Learning in terms of the theory and application of algorithms, it still faces with many difficulties in realizing the real application of Reinforcement Learning. It is increasingly difficult to meet the application requirements of current complex problems with traditional

Reinforcement Learning algorithms. Therefore, the application of Reinforcement Learning in multi-agent systems has received considerable attention in the industry. Many scholars also apply Reinforcement Learning to multi-agent systems. Littman, for example, has proposed the Maximal Minimal Q Learning Algorithm (1994) and the Friend-or-Foe-Q Algorithm (2001). The Maximal Minimal Q Learning Algorithm is mainly applied in the environment with only two agents, while in the Friend-or-Foe-Q algorithm, the agent has not only a cooperative relationship but also a competitive relationship in the algorithm. Hu and Wellman (2003) combined Nash balance in game theory with Q learning and proposed the Nash-Q algorithm. Tan (1993) proposed that for multi-agent systems, agents can share learning strategies with each other in a cooperative manner and interact with environmental information to accelerate the convergence rate. To solve the problem of conflict of interests between agents, Arai et al. (2000) proposed a method of profit distribution, which they successfully applied in a multi-agent system.

2.4.1 Trial and Error

The trial-and-error mechanism (Sutton and Barto, 1998) is a very important mechanism in Reinforcement Learning. In the early stages of learning, the system randomly selects actions to a large degree, learns from the environment, and makes judgments about the environment; when there is enough knowledge in the later stage of learning, the system selects actions based on system parameters, and less often makes a random selection. Thus, when the learning system does not have enough knowledge to make a good judgment of the environment, it has more to learn from the environment to optimize the system itself. It has more to learn from the environment to optimize the system itself; when the system

has accumulated some experience, it relies more on its judgement of the environment to select a better strategy. This is also the goal of Reinforcement Learning. The reason for random selection with a small probability is to prevent the system from falling into a local extremum and adapting to the changing environment based on existing learning. The core of this mechanism is to learn in trials and judge after learning. The more common trial and error mechanisms are Greedy strategies and Simulated Annealing:

Greedy strategy: When the random number rand $< 1 - \epsilon$, the optimal action is selected according to the system parameters; otherwise, the action is randomly selected with equal probability. When rand<1-ϵ, where rand is any random number on [0, 1] and s is the smaller real number of [0, 1], the optimal action is selected according to the system parameters; otherwise, the action is randomly selected with equal probability. The disadvantage of this algorithm is that ϵ is fixed in the learning iteration process, and the choice of late learning may fluctuate because of it.

Simulated Annealing: This mechanism evolved from metallurgy and is a simulation of metal annealing in thermodynamics, as shown in the following equation.

$$\mathrm{P(dE)} = \exp(\frac{dE}{kT})$$

where dE represents the energy difference that occurs at the temperature T, and $dE < 0$, k is a constant, and *exp* is a natural index. During the annealing process, $P(dE)$ becomes increasingly smaller as the temperature T decreases.

In the trial and error mechanism based on Simulated Annealing, the action and the new state j are randomly selected. When the random number $\mathrm{rand} < \mathrm{P(dE)}$, the new state j is accepted; otherwise, the best strategy is chosen. If you accept the new state to cool the temperature according to the annealing strategy, the common annealing strategy is shown as follows:

$$T_k = \frac{\lambda}{\log(k + k_0)}$$

$$T_k = \frac{\lambda}{1 + \mathrm{k}}$$

$$T_k = (1 - \frac{\lambda}{\mathrm{k}})T_0$$

$$T_k = \lambda^k T_0$$

where the parameter λ generally takes the real number in the interval (0, 1), T_k is the temperature at the kth iteration, k_0 is the initial value, and T_0 is the initial temperature. Reinforcement Learning has been widely used in the fields of robot learning, automatic control systems, game competition and dispatch management, as outlined in the next sections.

2.4.2 Control Systems

The inverted pendulum control system is a typical example of the application of Reinforcement Learning in control. It is a nonlinear unstable system. Many Reinforcement Learning articles use this control system as an experimental system for verifying Reinforcement Learning algorithms. When the inverted pendulum is balanced, it is rewarded. When the inverted pendulum fails, it is punished. For example, in the experimental system research of Barto et al. (1983) and Crites and Barto (1996), two

neurons, ASE and ACE, were learned through repeated experiments. The balance time of an inverted pendulum reaches several tens of minutes; Peng and Williams (1994) used the Q-learning algorithm to perform an experimental study on an inverted pendulum system and compared it to the AHC method.

Another area of application of Reinforcement Learning is in process control. The Reinforcement Learning method does not require a mathematical model of the external environment. Instead, it directly translates the performance indicators of the control system into an evaluation index. When the system performance indicators meet the requirements, the control actions are rewarded, otherwise they are punished. Through its own learning, the controller finally achieves the optimal control action. Even-Dar and Mansour (2003) gave an example of biological response control using Q-learning combined with a BP neural network; Klopf (1972) used the robot manipulator control with two degrees of freedom as an example to study the practical method of Reinforcement Learning in the adaptive control of nonlinear systems; Berenji and Khedkar (1992) implemented the motion evaluation random learning method using a kind of hardware to achieve the control task of the plug-in empty mission and the ball balancer.

2.4.3 Game Competitions

Game competition is an ongoing issue in the field of artificial intelligence in the field of artificial intelligence. Many scholars are also studying the application of Reinforcement Learning theory to game competitions. For instance, the earliest example of animal foraging games and other applications is the chess programme (Samuel 1959). More recently, Tesauro (1995) applied the instantaneous difference method to backgammon.

This is known as TD-Gammon. Backgammon has approximately 10^{20} states. Tesauro (1995) uses a three-layer BP neural network to associate the positions of the chess pieces on the chess player with the player's winning rate. Through training, he achieved only a negative record in 40 games.

2.4.4 Dispatch Management

Scheduling is an example of a stochastic optimal control problem with great economic value. Crites and Barto (1996) used the Reinforcement Learning algorithm in a four-elevator, 10-story system in which each elevator has its own position, direction, speed and a series of positions that indicate where the passenger is leaving. With more than 10^{22} state sets, such a system is difficult to manage with traditional dynamic programming methods (such as value iteration). Moreover, even if it takes only 1 second for each state to be traced back, it will take 1000 years for all the states in the retrospective set. In Crites and Barto (1996), the square of the average waiting time is used as the performance of the elevator scheduling algorithm and the anti-Chuan algorithm is used to train and present the neural network expressing the Q function. The Reinforcement Learning is superior to other algorithms.

2.4.5 Robotics

As an online learning method, Reinforcement Learning is currently applied most frequently, and is more appropriate for use in the field of robotics. In recent years, the field of applying Reinforcement Learning to the behaviour learning of intelligent robots has arisen in the world, including the learning of a single autonomous robot's behaviour and

the learning of multiple robot group behaviours. Maes and Brooks (1990) proposed a distributed learning mechanism to conduct behaviour selection learning in a walking study of “six-legged” robots. A set of behaviours and corresponding binary perceptual state sets were defined in advance and were reflected as perceptual states, and were reflected as perceptual states according to sensor information indicating either “0” or “1” status, and then the appropriate behaviour was selected. Makar et al. (2001) define three behaviour states in the task of learning the push box of a single mobile robot: finding box, pushing box, and dropping box in advance; they also define a condition for the behaviour to be transferred to another behaviour. The state is made up of sensors such as nanometres and infrared sensors. Because the state space is too large, a statistical clustering technique is used to represent similar states in groups, making the learning performance superior to the manual programming behavioural decision sequence. Santamaria and Ram (1997) proposed a case-based reasoning Reinforcement Learning adaptive reflection controller for the navigation control of mobile robots.

The application of Reinforcement Learning in multi-mobile robot systems is receiving increasing attention. Balch (1997) proposed that the Clay control structure be apprised to robot soccer matches. Unlike Reinforcement Learning based on behaviour control structures, Balch’s model combines Reinforcement Learning with motor schemas, ensuring the system has not only the adaptive learning ability of Reinforcement Learning but also the real-time performance of motor schemas. Michaud and Mataric (1998) used an improved Q-learning algorithm to implement four robots to execute the foraging task. Domain knowledge were used to transform the state space to action space, in order to map action space into conditional behaviour. As a result, the scale of the state space is

compressed and learning process is accelerated. Uchibe et al. (1996) use a Reinforcement Learning method for the division of no-learning areas and re-learning areas in his soccer robot learning to increase learning speed. There are also some EA reinforcement studies, such as the hill-climbing learning algorithm based on genetic algorithms proposed by Kamei et al. (2004), which use genetic algorithms to learn reflective control parameters for multi-machine autonomous navigation control.

2.5 The Exam Timetabling Problem

The study of the timetable problem has a history of nearly half a century, with most research having occurred in the 21st century. With the development of science and technology, many researchers have designed various solutions. These methods mainly include graph-based heuristics, local search-based techniques, population-based algorithms and Hyper-heuristic technologies.

2.5.1 Graph-based Heuristics

The graph colouring problem concerns how to assign colours to the vertices in a graph. The object of the examination schedule question is how to assign time slot for the exam. The two kinds of questions can be converted, that is, each test subject can be regarded as a vertex in the graph colouring problem. Some two tests with hard constraint conflicts can be expressed as one edge of the corresponding two vertices in the graph colouring problem. The graph colouring problem requires that the adjacent two vertices cannot be assigned the same colour, which is consistent with the requirements of the examination timetabling

problem that two exams with hard constraint conflicts cannot be scheduled at the same time. The above theories were first proposed in 1967 by Welsh and Powell, who constructed a reasonable bridge for graph colouring and exam scheduling problems. They referred to the corresponding constraints in the graph colouring problem and proposed different soft constraints to evaluate the quality of the results. Carter and Laporte et al. (1996) first proposed five sorting strategies for generating test schedules. The authors pointed out that the largest clusters, that is, each vertex in a subgraph that are connected to other vertices, can be constructed to the initial solution by the graph colouring heuristic method and backtracking method. For the timetabling problem, the maximum cluster size theoretically determines the minimum period of the scheduling problem. In addition, another major contribution of the paper to the study of timetable issues is the presentation of a set of 13 questions specific to the examination timetabling problem known as the University of Toronto data. This set of questions has been used as a standard test question to date.

In 1998, Burke and Newall et al. proposed a stochastic strategy and designed two different selection strategies for exam scheduling problems: tournament selection strategy and preference selection strategy. In the experiment, the authors combined the random strategy with the previously proposed graph heuristic method and used the two selection strategies and the Toronto data as the performance test algorithm to test the problem. The final result shows that the algorithm performs well. Burke and Newall (2004) also proposed an adaptive sorting strategy to dynamically control the order of test subjects in the iterative process. That same year, Asmuni and Burke et al. (2004) proposed a fuzzy logic method to control the order of exams. In this method, different ambiguity functions are used for

different test questions. After the difficulty of arranging test subjects is estimated, the order of the test to be scheduled is adjusted.

The graph heuristic method is still widely used to deal with examination timetabling problems. In its current form, it is different from the simple graph heuristic approach of the early days. The current research trend is to combine graph heuristics with other techniques to form a kind of hybrid approach, which derived the so-called graph-based Hyper-heuristic technology, combines graph heuristics with Hyper-heuristic, and thus has become one of the classic methods for solving exam scheduling problems.

2.5.2 Local Search Metaheuristics

Local search technology (Paquete, Stützle 2002a) is a metaheuristic method because it can handle different constraints relatively easily and is widely used in the field of examination timetabling. As a general term for a search method, it is the current solution for in-depth and detailed iterative searches of the neighbourhood, in order to find a high-quality solution. Different neighbourhood construction strategies and different mobile operators constitute various local search techniques.

The Tabu search algorithm is a representative local search method. The original idea was proposed by Glover (1989). The algorithm starts with an initial solution to the problem and searches for the optimal solution based on the specific search direction. During the search process, the Tabu list plays a key role in the search. In the Tabu table, a series of taboo criteria are stored. Within a certain step range, the algorithm is prohibited from performing a search similar to the Tabu criterion, thereby preventing the algorithm from being repeatedly searched during the search process. Di Gaspero and Schaerf (2000)

proposed a class of Tabu search algorithms in order to ascertain the impact of neighbourhood settings on violations of hard and soft constraints. They used preference selection and no preference selection strategies in algorithm design to build adaptive cost functions and dynamically controlled the length of taboo lists in the iterative process. Finally, the authors proved that the adaptive cost function can select the proper neighbourhood and play a key role in avoiding violation of the constraint.

Further improvements to the above method were reported in Di Gaspero and Schaerf (2002). This improved method will change the time slot of a certain exam or exchange the time slot of a certain number of exams as a new neighbourhood so that the algorithm forms multiple neighbourhoods during the search process. Finally, through this neighbourhood search method, the results obtained are improved. White and Xie (2000) designed a four-level Tabu search method for exam timetabling problems, and the quality of other solutions can be improved through different levels of search. Paquete and Stützle (2002b) proposed a Tabu search strategy for sorting priorities. In the running process of the algorithm, the length of the taboo linked list was adaptively set according to the violation of the solution.

2.5.3 Population-based Metaheuristics

As a typical representative of evolutionary algorithms, genetic algorithms have extensive research foundations in the field of examination timelines. In Corne and Ross et al. (1994), a brief summary of the various genetic algorithms for solving educational timetables was presented. The authors pointed out that for some problems with graph colouring as a template and a specific structure, if a direct encoding method is used, the genetic algorithm

does not facilitate resolution of the problem. Ross et al. (1995, 1997), who studied the reason why direct coding was not suitable for timetabling and graph colouring problems, suggested that genetic algorithms should be used to find suitable algorithms that can solve a variety of specific problems instead of directly finding all actual solutions to problems. Terashima-Marin and Ross et al. (1999) designed a special kind of cluster-based crossover operator for the timetable problem that used different reorganization strategies to breed the population. Eventually, they achieved the same research as Ross and other scholars. The conclusion is that the direct coding method is not suitable for using the genetic algorithm to solve the timetabling problem.

Erben (2000) studied the coding method, crossover operator and mutation operator, and fitness function in the genetic algorithm, and proposed a grouping genetic algorithm. Compared with other methods at the time in the literature, his method does not provide the best results, but the advantage of this method is that its calculation time is very short. Wong and Côté et al. (2002) identified a genetic algorithm to solve the examination timetabling problem, which uses the championship selection method to select the parent and merge the repair strategy with the mutation strategy to generate a better candidate solution. Côté and Wong et al. (2004) set the minimum test period (length of the timetable) and the minimum interval conflict as two separate goals, establishing the test schedule as a goal optimization problem and designing the corresponding objective evolutionary algorithm. Ülker and Özcan et al. (2006) used precedence coding as the representation method, combined with different crossover operators, and finally designed a new genetic algorithm to deal with graph colouring problems and exam timetabling problems.

2.5.4 The Hyper-Heuristic Method

The Hyper-heuristic method is a popular technology that has emerged in recent years. Unlike the traditional approach of searching for the right solution using only heuristic methods, it can be seen as a heuristic algorithm to find the right heuristic algorithm. Ross et al. (1997) discussed observations of exam timetabling problems of using GA based approaches.

Ahmadi et al. (2003) proposed new ideas for resolving problems related to the examination schedule. They used a variable neighbourhood search to find combinations of various heuristic methods to solve different exam schedule problems. Ross et al. (2004) proposed a steady-state genetic algorithm to solve the problems of curriculum scheduling and exam scheduling. In this algorithm, the search target of the genetic algorithm is to find a suitable heuristic algorithm.

2.6 Summary

In this chapter, the fundamentals and details of the solution techniques precisely proposed for the examination timetabling was discussed, including metaheuristics, Hyper-heuristic and Reinforcement Learning. Two metaheuristic algorithms were presented in more detail, Squeaky Wheel and Simulated Annealing, which are often used for solving combination optimization problems. Through an extensive review of the literature, the framework for Hyper-heuristics and online learning Hyper-heuristics were covered. With regard to Reinforcement Learning, the trial- error mechanism was introduced, together with several examples (Maes, Brooks 1990; Santamaria, Ram 1997; Balch, 1997; Michaud, Mataric

1998; Makar et al., 2001) to show the range of its applications. A detailed literature review of exam timetabling problems, the focus of the current research, also was presented. The review included various types of research approaches that have been implemented previously.

This research is mainly using hyper-heuristic and metaheuristic to conduct the search for the exam timetabling problems. The metaheuristics are, as analysed in this chapter, rather problem independent and therefore more promising in finding the global optimal compared to heuristics. The focus of this research is to design high problem independent approaches. As a two-level general framework, hyper-heuristic has also been bringing into this research out of its' generality.

In the next chapter, exam timetabling problems will be formally introduced in detail along with the ITC2007 benchmark that is used in this research. The features of the ITC2007 benchmarks, including constraints and the general solving process of the exam timetabling problems, will be all provide.

CHAPTER 3. EXAM TIMETABLING PROBLEMS

3.1 Introduction

In this chapter, the focus of the research reported in this thesis, exam timetabling problems (ETP), are introduced in detail. ETP are among the most popular scheduling and optimization problems that have been studied in the last decade. ETP arise in real life and are commonplace in almost all of the academic institutions in the world. The difficulty associated with ETP arises mainly out of the number of students and exams affected, as well as the difficult constraints which vary between faculties and institutions. For these reasons, ETP have been selected as the target case study for the proposed general approaches described in this thesis.

This chapter includes the background to the problem of exam scheduling, demonstrations of the construction and optimization stages of ETP as well as the currently available benchmarks for ETP. In addition, this chapter includes a detailed introduction to and definition of the ITC2007 benchmark which is used for the test in this thesis, including the hard constraints and soft constraints. This chapter consists of four main sections: the introduction to the problem, demonstrations of the classic approaches, the benchmarks described in the literature and a detailed definition of the ITC2007 benchmark.

3.2 Exam Timetabling Problems

Exam timetabling problems (ETP) (Burke et al., 2006) can be considered as having a certain number of examinations that must be arranged within a set period of time. Because the scheduling problem has a very wide application prospect in the real world, and there are usually a large number of constraints for different schedule planning problems, as well as a lot of conflicting relationships among these constraints, exam scheduling has always been a difficult problem and a popular issue in the field of resource scheduling problems.

The constraints involved are usually divided into two categories: hard and soft. Hard constraints are not supposed to be violated in order to generate a feasible solution. For example, one student cannot be assigned with two exams in the same period. Or, for all the exams allocated with the same room at the same time, the overall number of students of all these exams cannot exceed the room capacity.

Soft constraints are different from the hard constraints in that they can be violated when arranging exams. Finding a feasible result that satisfies all soft constraints is almost impossible. Soft constraints have greater uncertainty and can be set according to the special external objective environment and practical needs. In related studies on examination timetable issues, it is usually possible to arrange conflict examination subjects more evenly throughout the examination cycle as one of the important soft constraints, so that students have sufficient time for revision before each of the examinations they need to take. Soft constraints are generally used for quality inspection of a given scheduling solution.

3.3 Construction and Optimization

Over the past few years, many methods have emerged that can be applied to the timetabling problem. In this chapter, two stages of these methods are discussed: construction and optimization. To present the construction process when solving the examination timetabling problems, the literature on graph colouring methods (Bondy and Murty, 1976) will be reviewed. To demonstrate the optimization process of the timetabling problem (Burke et al., 2004), the literature on the Adaption of Heuristic Orderings will be discussed.

3.3.1 Construction

The academic timetabling problem is a typical timetabling problem, and the discussion of the timetabling problem is about the timetabling of university exams. In (Bondy and Murty, 1976, to improves the scientific level and rationality of exam schedules, the graph colouring method is used to solve the exam scheduling problem and make the algorithm meet two needs. Firstly, by arranging exam time slots rationally, the number of people involved in the conflict is minimised to a minimum. Secondly, most students' test intervals are as large as possible to ensure that students have enough rest time.

The problem of university examination timetabling is to arrange examinations in a limited time period while avoiding conflicts and satisfying certain constraints. The applicable hard and soft constraints in (Bondy and Murty, 1976) are listed as follows:

Hard constraints

1. The test period is fixed, usually 4 test periods per day.
2. All professional examinations in the school are arranged within one week.

3. No student is allowed to have more than one exam scheduled at a given time.
4. The examination room is limited, and the invigilator is limited. Both of these are requirements.

Soft constraints

1. The number of test subjects arranged in each time period should be as balanced as possible.
2. There should be a certain gap between examination courses for students of the same grade.
3. The number of clashes between test takers should be minimized.

3.3.1.1 Establishing the graph colouring model

Suppose there are *n* courses in the exam timetable in total. The exams are scheduled to take place in *m* periods. The number of exams for each course varies. The construction map G contains *n* vertices and each vertex represents a course. The weight of each vertex represents the total number of students enrolled in the class, and the weights on the sides represent the number of students enrolled in the course.

When constructing an undirected graph, first select a major, then extract the courses learned by the various grades of the majority in this semester to form an undirected complete subgraph (also referred to as a group) and form the number of the weight of the vertices in each cluster based on the number of students.

Examination Question Description

Suppose a department has three grades who must perform the final exam at the same time, assuming no consideration of make-up exams. There are 50 students in the first grade registered with four subjects for testing. Here the first-grade subjects are referred to as ABCD. There are four second grade subjects (EFGH), involving 60 students. In addition, there are two subjects at the third grade, I and J, involving 40 students. From a total of 150 students in all grades from all grades, 15 students in the second grade need to take remedial courses C, another 5 students need to take remedial courses A, and 8 of the candidates in third grade are required to do retakes in course B. Another 15 students in the second grade must not only participate in exams for courses E, F, G and H, but also take part in the exam for course C. It is necessary to connect C with the vertices E, F, G, H in the group and add the weight 15 to the corresponding edge. Other connection methods for make-up exams and the set of weights are the same as above, so that links between the original orphaned groups are established, as shown in Figure 5.

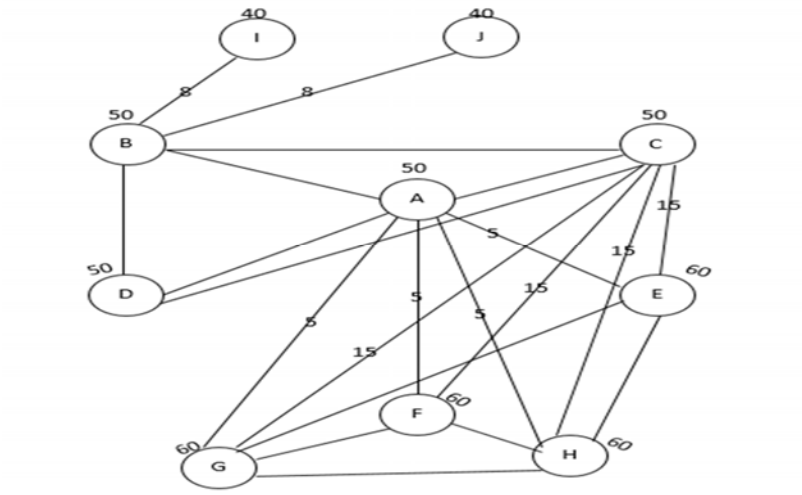

Figure 5:Model of the timetabling problem (Bondy and Murty, 1976)

For the specific exam questions, we give a custom concept: the original group. The original group mentioned in the following text refers to a group of exam subjects in each grade.

In this way, there are three groups of ACEFGH, ABCD, IJB in Figure 5. The largest group, ACEFGH, is based on the original group EFGH (second-year test subjects), because some individuals need to make up grade courses A and C, connecting A and C together. Consequently, neither A nor C can be in the same time period as any course from the group comprised of EFGH. The group IJB is based on the original group U (third-year exam subjects), because some individuals need to make up other grade courses B. After the formation of links, it is clear that the phenomenon of make-up examinations has increased the complexity of the test course schedule. The public class and make-up exams are similar in nature and are not elaborated here. The weight of each vertex in Figure 5 indicates the total number of students enrolled in the vertex course, and the weight on the side indicates the number of students enrolled in the course, represented by the two vertices at the same time.

In general, in the test arrangement, the time period provided for the test should not be less than the maximum number of vertices of the largest original group. The examination subjects should be arranged first so that the examination time between the subjects is as close as possible. When arranging the supplementary examination subjects (which are also examination subjects for others), it is necessary to consider not only avoiding conflicts, but also is the need for a certain gap between the time periods.

3.3.1.2 Workflow of this method

To construct an exam timetable, all the exams are represented as vertices and the

scheduling process is the colouring process of all vertices. The graph colouring approach in timetabling is sequential. All the exams are scheduled/coloured by order, which is decided by a graph colouring algorithm. The constraints are satisfied during this process. The side links two vertices represent are the constraints which ensure the linked two exams will not be scheduled in the same timeslot.

There are five main graph colouring heuristics that can be used to determine the scheduling sequence of exams during the construction process (Burke, 2007): least saturation degree first (SD), largest colour degree first (CD), largest degree first (LD), largest enrolment first (LE), largest weighted degree first (LWD).

The colour value in the algorithm corresponds to the actual time period. In order to ensure that courses of the same grade or even the same student are separated in time, attention must be paid to the selection of colour when colouring. Therefore, when each vertex in the same cluster is shaded, it is done according to a certain algorithm.

When all vertices in a cluster have not yet been coloured, the number *m* of colours provided may either be all available for the current regimen, or they may be partially available. If all the colours are available, the available space for the vertices in the cluster is very large. At this time, the vertices in the cluster should be coloured as evenly spaced as possible. The method for generating the interval is as follows:

1. The current total number of colours is *m*. There is a total of *vnum* vertices in the cluster. First, the first vertex in the cluster is coloured as *l*, denoted as $k = l$, and let *vnum*--;
2. Determine whether there are any uncoloured points in the current group. If so, go to step 3, and if not, stop;
3. Calculate $k+ = (m-k)/vnum$; make the current vertex coloured as *k*. Let *vnum*-- ; go to

step 2.

For example, the total number of colours provided is m=16, and 1-16 are colour codes. As shown in Figure 5, the current to-be-coloured group is Exam = {ABCDEFGHIJ}. To demonstrate the scheduling process, we simply calculate the degree value of the vertices (exams) as the number of registered students plus the side weight link to the current vertex (the number of students taking both exams the vertices represented).

The degree of all the exams in this scenario is shown in Figure 6:

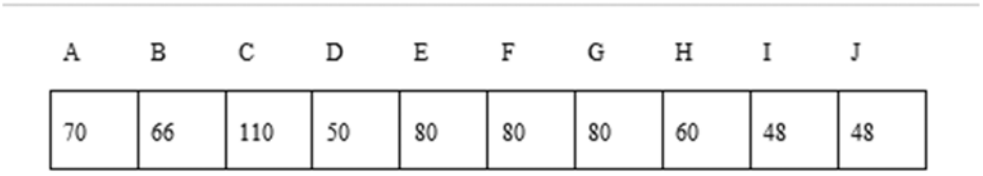

Figure 6:Exam degree values

If 8 colours are available for scheduling (i.e. if there are 8 available time slots), the colour code would be 1-8, as shown in Figure 7.

Figure 7:Colouring Table provided 16 kinds of colours

If using the largest degree graph colouring algorithm, the colouring process can be described as shown in Figure 8.

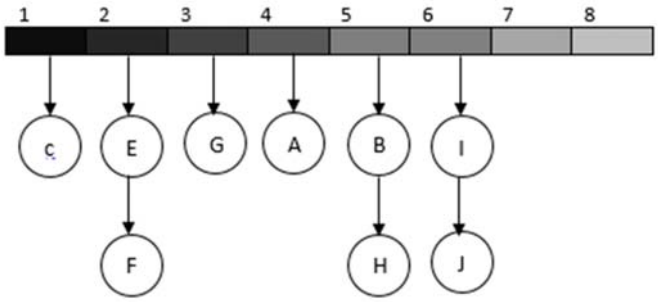

Figure 8: Colouring Table after Colouring

In this way, the vertices in the group Exam are all coloured. If the number of colours is only enough in sequence, then the algorithm will check the availability of the previous allocated colour, then the available colour will be allocated for the reset of vertices. Therefore, an algorithm is demonstrated.

As can be seen from the process above, the ordering of the exams to be scheduled affected the final result of scheduling. Therefore, the choice of the graph colouring heuristic is critical. Also, under certain circumstances, the construction could fail when using the graph colouring heuristics. Therefore, in this research, the Squeaky Wheel Optimization is used to construct the initial solution. The preceding section has been included only to describe the construction process of scheduling.

3.3.2 Optimization

In a paper on the adaption of heuristic orderings as a means of solving exam timetabling

problems, Burke and Newell (2004) concluded that five kinds of soft constraints that will affect the exam timetabling problems. These are shown in Table 1.

Table 1:Exam Timetabling Problem Soft Constraints in (Bondy and Murty, 1976)

Number	Constraints
1	One student cannot attend two exams at the same time
2	One period cannot be allocated more seats than are available
3	The time between exams taken by the same students should be reasonable.
4	First-order conflicts describe situations in which an exam is allocated to the same period.
5	Second-order conflicts, two conflicting exams cannot be scheduled in the same period

Next, some notations are made to evaluate the solution and compare it with other methods.

$$t_{ip} = 1 \; if \; exam \; i \; is \; scheduled \; in \; period \; p, 0 \; otherwise$$
$$c_{ij} \; is \; the \; number \; of \; students \; taking \; both \; exams \; i \; and \; j$$
$$d_{pq} = 3 \; if \; period \; p \; is \; on \; the \; same \; day \; as \; period \; q;$$
$$= \; 1 \; if \; they \; are \; on \; adjancent \; days; 0 \; otherwise.$$
$$s_i \; is \; the \; number \; of \; students \; taking \; exam \; i$$

Also, there are two Equations:

$$\sum_{i=1}^{E-1} \sum_{j=i}^{E} \sum_{p=1}^{P-5} \sum_{q=p+1}^{P} 2^{5-q+p} t_{ip} t_{jq} c_{ij} + \sum_{i=1}^{E} 50000 t_{i(P+1)}, \quad (1)$$

$$\sum_{i=1}^{E-1} \sum_{j=i+1}^{E} \left[\sum_{p=1}^{P} t_{ip} t_{j(p+1)} c_{ij} d_{p(p+1)} + t_{ip} t_{j(p-1)} c_{ij} d_{p(p-1)} \right] + \sum_{i=1}^{E} 5000 t_{i(P+1)}. \quad (2)$$

Equation (1) is used for measuring the spread of students. Equation (2) can be used

to represent students who take two exams and is not satisfied with the timetable. The measurement of the schedule is to weight the number of unscheduled exams by 5000. Next, the hard constraints must be regarded as equal in the following three Equations.

$$\sum_{p=1}^{P+1} t_{ip} = 1, \quad \forall i \in \{1, \ldots, E\}, \tag{3}$$

$$\sum_{i=1}^{E-1} \sum_{j=i+1}^{E} \sum_{p=1}^{P} t_{ip} t_{jp} c_{ij} = 0, \tag{4}$$

$$\sum_{i=1}^{E} t_{ip} s_i \leqslant S, \quad \forall p \in \{1, \ldots, P\}. \tag{5}$$

where Equation (3) shows that each event can only be scheduled once, and Equation(4) shows that at the same time there are no conflicting events. Equation (5) shows the constraint of the maximum number of seats available. In (Bondy and Murty, 1976), the most difficult exam is re-scheduled at each stage and the difficulty of all the exams are updated during the whole optimisation process.

3.4 Benchmark Datasets

3.4.1 Toronto, Nottingham and Melbourne Datasets

Over the past few decades, a lot of benchmarking has been put forward to solve exam timetabling problems. Qu, Burke, McCollum et al. (2006) covered several kinds of benchmarking in her survey of the exam timetabling problems in 2006. Qu's introduction includes University of Toronto Benchmark Data, University of Nottingham Benchmark

Data and University of Melbourne Benchmark Data.

The University of Toronto Benchmark Data has been tested on a wide range of actual timesheet benchmarks to evaluate the scheduling algorithm. The data is drawn from a set of 13 real-world exam timetabling problems in 11 schools, including three Canadian high schools, five Canadian universities, one American university, one British university and one university in Saudi Arabia. This method constructs a Conflict Matrix C:

$$c_{ij} = 1 \; when \; two \; exam \; are \; conflicted$$
$$c_{ij} = 0 \; otherwise$$

This method defines the term Conflict Density, which represents the ratio between the numbers of elements of value "1" to the total number of elements in the conflict matrix. The Toronto Benchmark Datasets (Di Gaspero et al., 2007), shown in Table 2. Also, there are four variants of the Toronto Benchmark Datasets, shown in Table 3.

Table 2: Characteristics of the Toronto Dataset

Problem Instance	Exams	Students	Enrolments	Conflict Density	Timeslots
car91 I	682	16925	56877	0.13	35
car91 II	682	16925	56242/56877	0.13	35
car92 I	543	18419	55522	0.14	32
car92 II	543	18419	55189/55522	0.14	32
ear83 I	190	1125	8109	0.27	24
ear83 II	189	1108	8014	0.27	24
hec92 I	81	2823	10632	0.42	18
hec92 II	80	2823	10625	0.42	18
kfu93	461	5349	25113	0.06	20
lse91	381	2726	10918	0.06	18
pur93 I	2419	30032	120681	0.03	42

pur93 II	2419	30032	120686/120681	0.03	42
rye92	486	11483	45051	0.07	23
sta83 I	139	611	5751	0.14	13
sta83 II	138	549	5689	0.14	13
tre92	261	4360	14901	0.06	23
uta92 I	622	21266	58979	0.13	35
uta92 II	638	21329	59144	0.13	35
ute92	184	2749	11793	0.08	10
yor83 I	181	941	6034	0.20	21
yor83 II	180	919	6012	0.29	21

Table 3:Toronto Dataset Variants

	Variants	Objectives
Toronto a	graph colouring	to minimize the number of timeslots needed
Toronto b	capacitated with cost	to space out conflicting exams within limited (a fixed number of) timeslots
Toronto c	capacitated with a modified cost	same as above, and to minimize number of students sitting two exams overnight
Toronto d	estimated capacity and timeslots	to minimize students sitting two adjacent exams on the same day

The characteristics of the University of Nottingham Benchmark Data, which was proposed by Burke, Newall and Weare (1996), are shown in Table 4. For these problems, there is a limit to the number of students who can take the examination during each period. The penalty function is calculated by calculating the number of cases in which there are two examinations (weighted to 3) in the adjacent period of the same day, and the number of times that a student has two exams but two days apart during adjacent periods. Any unplanned examination will incur further penalties of 5000 people. Weights are optional, although they are designed to reflect the relative importance of constraints. In the context of our problem definition, we will minimize the expression, subject to all constraints.

Table 4:Nottingham Benchmark Datasets

	Nottingham a	**Nottingham b**
Exams	800	800
Students	7896	7896
Timeslots	23,26	23
Enrolments	34265	34265
Conflicts	10034	10034
Capacity	1550	1550
Density	0.03	0.03
Objective	minimize adjacent exams on the same day	minimize adjacent exams on the same day and overnight

The University of Melbourne Benchmark Data was proposed by Merlot et al. (2002). Its characteristics are shown in Table 5.

Table 5:Melbourne Benchmark Datasets

	Exams	**Timeslots**	**Students**	**Enrolment**	**Objective**
I	521	28	20656	62248	minimize adjacent exams on the same day or overnight
II	562	31	19816	60637	same as above

Schaerf and Di Gaspero (2001) also use these questions to test Tabu search techniques. Burke and Newall (1999) use a subset of these questions to assess the impact of splitting the problem into smaller chunks. Because of the use of this free data, we can directly compare these three studies and where appropriate, give a time period for testing. Where appropriate, give a time period for testing. Two methods are used to evaluate the quality of the solution. One of them is introduced by Carter, LaPorte and Lee (1996), which

assigned close to cost WS whenever students must separate two examinations. These weights are defined as W1 = 16, W2 = 8, W3 = 4, W4 = 2, W5 = 1. Then divide the sum of all these measures by the number of students, giving each student an average penalty. For these problems, there is no limit to the number of seats in each stage. In our previous question definition, Carter, LaPorte and Lee (1996)'s approach involves minimizing the expression and ignoring the seat limit requirement for expression.

3.4.2 ITC2007 Datasets

The Second International Timetable Competition (ITC2007) is a dataset divided into exam scheduling and course scheduling. This dataset must be used to focus on checking the dataset. To ensure that researchers can evaluate their algorithms based on real-world timetabling problems, McCollum et al. (2007) create a platform for ITC2007 purposes. ITC2007 checks that the dataset contains very few hard constraints. First, no student can sit more than one exam at the same time. Second, the number of students participating in the examination must not exceed the room capacity. Third, the test to be added to the time slot shall not exceed the length of the time slot. Fourth, there is a need to meet some specific requirements, such as exam A to be assigned after test B, or test A must be in Room 15 and so on. Other requirements may be to reduce second-level conflict on the same day, reduce the duration of the test in one time period, reduce the specific time slots or space used, and hold large checks as early as possible for the purpose of the dataset. (See McCollum and McMullan et al., 2007 for details.)

The dataset was studied by iterative forward search, hill climbing and great deluge algorithm. GRASP, Simulated Annealing and mathematical programming are used in

multi-stage methods to solve ITC2007 datasets. Qu and Burke et al. (2009) uses the two-stage method and adaptive heuristic ordering as the construction phase and enhances the solution by using the extended great deluge algorithm (Qu et al., 2009). Table 6 shows the details of for the ITC2007 dataset.

Table 6: Characteristics of the ITC2007 Benchmark Problem Instances

Instance	Conflict Density %	Exams	Students	Periods	Rooms	Period HC	Room HC
Exam 1	5.05	607	7891	54	7	12	0
Exam 2	1.17	870	12743	40	49	12	2
Exam 3	2.62	934	16439	36	48	170	15
Exam 4	15	273	5045	21	1	40	0
Exam 5	0.87	1018	9253	42	3	27	0
Exam 6	6.16	242	7909	16	8	23	0
Exam 7	1.93	1096	14676	80	15	28	0
Exam 8	4.55	598	7718	80	8	20	1
Exam 9	7.48	169	655	25	3	10	0
Exam 10	4.97	214	1577	32	48	58	0
Exam 11	2.62	934	16439	26	40	170	15
Exam 12	18.45	78	1653	12	50	9	7

For exam timetabling problems, the ITC2007 gives several rules for participants to comply with. It is mainly retrieved from ITC2002 but makes some modifications.

3.5 ITC2007 Problem Definition

In this thesis, the proposed frameworks were applied and tested using the ITC2007 benchmarks. The ITC2007 benchmark consists of 12 datasets extracted from real-world timetabling problems. The 12 datasets feature rather differed characteristics, which is consistent with the general approach taken by this thesis. The proposed approach would be able to learn and adapt when facing different datasets from the ITC2007 benchmarks. Also,

the ITC2007 is one of the most popular benchmarks in the literature, generating enough published results for comparison with the research reported in this thesis.

3.5.1 Hard Constraints

Hard constraints must not be violated during the schedule. For example, a conflict test (that is, multiple test subjects that the same student is required to attend) must not be scheduled in the same time period. The number of participants in a certain test must not exceed the capacity of the classroom. Normally, a schedule that meets a hard constraint is called a feasible solution.

Table 7:ITC2007 Hard Constraints

No.	Hard Constraints
H1	A student cannot sit more than one exam at the same time.
H2	The capacity of the exam should not exceed the room capacity.
H3	The exam length should not violate the timeslot length.
H4	The sequence or order of an exam must be respected.
H5	Schedule exam into the specified room (room-related hard constraints).

3.5.2 Soft Constraints

The soft constraints are different from hard constraints as they can be violated when arranging examinations. Typically, finding a feasible result that satisfies all soft constraints is almost impossible. Soft constraints have greater uncertainty and can be set according to the special external objective environment and practical needs. In related studies on examination timetable issues, it is usually possible to arrange conflict examination subjects

more evenly throughout the examination cycle as one of the important soft constraints, so that students have sufficient time to review before a number of examinations they want to participate in. Another common soft constraint is to place large exams as early as possible in the exam cycle so that teachers can have sufficient time for marking. Soft constraints general index as a judge exam schedule, and quality inspection schedule by estimating a violation of the soft constraints imposed on the overall schedule.

Table 8:ITC2007 Soft Constraints

No.	Soft Constraints
S1	Two exams in a row: minimize student sitting consecutive exams on the same day.
S2	Two exams in a day: minimize student sitting more than two exams in a day.
S3	Spread of exams: Each set of student examinations should be spread as evenly as possible over the exam period.
S4	Mixed duration: minimize the number of exams with a different duration that is scheduled in the same room.
S5	Larger examination schedule late in the timetable: minimize the number of large exams that appear 'late' in the timetable.
S6	Period penalty: minimize the number of exams scheduled in the period with a penalty.
S7	Room penalty: minimize the number of exams scheduled in a room with a penalty.

3.5.3 Instances and File Formats

In ITC2007, there are 21 exam and course datasets available which are all retrieved from the University of Udine. Each instance includes a file with a header and four sections: courses, rooms, curricula and constraints. The input and output file formats are listed in Appendix A.

3.6 Summary

In this chapter, we introduced a classic optimization problem: exam timetabling. We used exam timetabling problems as a case study in this research. The construction and optimization processes were all demonstrated by the classic approaches described in the literature. In addition, all the popular benchmarks used in the research area of scheduling were presented with a detailed analysis of the ITC2007 benchmarks. The implementation and testing of the proposed approaches in this thesis are based on the ITC2007 benchmark.

The exam timetabling problems is a classic, yet unique, scheduling problem. Although the research of the exam timetabling problems can be dated to the 90's, the updated benchmarks have been improved the difficulty and complexity of the exam timetabling problems. And the works in the literature are focused more on getting the best global result instead of improving the approaches' generality (section 2.5). This research proposed two general systems based on an up to date benchmark ITC2007. It is believed that through solving the exam timetabling problems in a more general way, the proposed approaches in this research can be used in other scheduling and optimisation problems.

In the next chapter, an online learning Hyper-heuristic framework based on Simulated Annealing and various Reinforcement Learning algorithms will be proposed and introduced in detail. A Reinforcement Learning Simulated Annealing approach also will be proposed. For this proposed framework, designs of multiple heuristic selection methods and utility update methods will be presented.

CHAPTER 4. ONLINE LEARNING HYPER-HEURISTICS

4.1 Introduction

Previous studies show that a combination of different selection Hyper-heuristic components yields different performances (Özcan et al., 2008). To further improve the generality of the hyper-heuristics, this study investigates the performance of a set of selection Hyper-heuristics combining different Reinforcement Learning schemes for heuristic/operator selection and move acceptance methods for examination timetabling.

In this chapter, a single-point online learning perturbative hyper-heuristic approach is proposed with a case study on the defined problem formula from the ITC2007 benchmark. This research is a Hyper-heuristic method which uses Reinforcement Learning methods combined with a Simulated Annealing algorithm in the high-level heuristic selection stage. This proposed approach is named RL-SA-HH.

The first section describes the construction process using Squeaky Wheel Optimization methodology. The second section describes the proposed optimization Hyper-heuristic approach components, including the heuristic selection, the move acceptance and the designation of the low-level heuristics. As one of the main focus of this research is the proposed RL-SA-HH, the embedded successful Reinforcement Learning techniques are explained. The use of Reinforcement Learning algorithms in the Hyper-heuristic is described in detail. This is followed by a discussion of the contribution of the implemented model as well as a direction for future research.

4.2 Reinforcement Learning in Examination Timetabling

4.2.1 Motivation

The benefit of using the learning method to solve the scheduling problem is that the learned agent will adapt to the problem (internal model of the problem domain). Even when the problem changes slightly, for example, due to changes in student number or room space, the agent can resolve the new problem by itself without much human intervention. This is a very useful feature in real-world applications. The research in the last decade in the academic scheduling and optimization field has achieved great success and thus more attention is now focused on using these achievements to solve real-world scheduling and optimization problems. Real-world problems, compared to the academic problems, feature higher complexity and more flexibility. Therefore, a premise of this research is that learning ability is vital, and so the use of Reinforcement Learning approaches is necessary.

Another potential advantage of using a Reinforcement Learning method is that it can find a good compromise solution when part of the scheduling has been arranged. For example, imagine a situation in which half the examinations have taken place and the administrator finds that the rest of the exam schedule must be re-arranged. In this situation, a sophisticated learned agent nevertheless can find the optimal solution for the remaining exams no matter how good or bad the previous exams have been. Thus, even when using the learning method to achieve a solution may require a more computational resource, it still has good flexibility compared to other dedicated problem-specific scheduling methods.

4.2.2 Reinforcement Learning

To allow Reinforcement Learning to be used to solve exam timetabling problems, these problems must be formulated into a Reinforcement Learning problem for the Reinforcement Learning to solve. The learning process of Reinforcement Learning always requires building a model consisting of state, action, value and reward function (Sutton and Barto, 1998). The learning process could be followed by a different policy.

In exam timetabling problems, the state space could be considered to be a mapping consisting of students, exams, rooms and periods.

State Space:

(Students status) X (exams status) X (timeslots status) ->S

The action space would be the schedule of the exams in different rooms and periods.

Action Space:

(Adjust students) X (adjust exams) X (adjust timeslots) ->A

Reward function here can take the evaluation of the exam solutions into consideration.

The research on using Reinforcement Learning to solve the exam timetabling problems in this research uses the ITC2007 benchmarks. As discussed in Chapter 3, the benchmark is rather difficult and complicated. Observation of the datasets in the ITC2007 benchmark shows that both the state space and the actions space would be very large with the formula given above. Therefore, using Reinforcement Learning algorithms to solve to solve the exam timetabling problems in the ITC2007 benchmark would be rather computational and time-consuming. It is not difficult to imagine that the <State, Action>

map would be even bigger. Also, the transition probability function is unknown. Given the above conditions, although using Reinforcement Learning algorithms to solve the exam timetabling problems is not impossible, the learning process could be very slow and hard to direct.

4.2.3 Combination-based Reinforcement Learning for Exam Timetabling

As described above, straightforward Reinforcement Learning would be difficult to utilize in solving timetabling problems and the training process would be difficult to control and rather slow.

Therefore, combining Reinforcement Learning with another technique is more realistic. In the literature, it has been proposed to combine learning algorithms with different techniques to solve the exam timetabling problems (Burke and Hyde et al., 2010). Of these techniques, the focus of the research on Hyper-heuristic algorithms has been more on the learning process. The Hyper-heuristic with learning process has been proven to be promising during the last decade in the literature (Burke and Kendall et al., 2003). Özcan and Misir et al. (2012) combined Reinforcement Learning concept with greedy deluge techniques into a Hyper-heuristic to solve exam timetabling problems.

Thus, in this research, to further investigate and improve the learning process, various policies as well as reward functions, are introduced. The Hyper-heuristic searching process is modeled as a Reinforcement Learning process. The Reinforcement Learning process consists of the policy, actions, reward function and value function. In the Hyper-heuristic solving process, the learning policy is the heuristic selection method. The low-level heuristics are the actions. Therefore, the reward function in Reinforcement Learning is the

reward function for the low-level heuristics. The value function is used to represent the value of the <state, action>. It is not only the immediate reward, but the long-term value used to improve the policy. This can be considered as the utility value function in Reinforcement Learning. In this research, different value functions that include the estimated reward are also implemented.

These techniques used in Reinforcement Learning algorithms have proved promising with the experimental results reported here. The tuning of the parameter that affects the learning process should be the focus of future work. Also, the two most popular techniques of Reinforcement Learning, Q-learning and SARSA (Chapter 2, Sutton Barto, 1998), are both able to combine with the Hyper-heuristic framework to define whether on-policy or off-policy can perform better with the Hyper-heuristic.

4.3 Squeaky Wheel Construction

In the research on the approach proposed in this chapter, the targeted Datasets use the ITC2007. Generally, this problem can be considered as a schedule of a set of events E, which are the exams, in the suitable set of timeslots T, which are the combination of a set of rooms R and a set of periods P with no violation of the hard constraints, as well as minimal violations of the soft constraints.

Listed below are the notations that must be noted:

- ei is a collection of n examinations (i = 1…, N)
- T is the number of timeslots.
- R is the number of rooms
- P is the number of periods

As described above, the proposed Hyper-heuristic is a single-point, perturbative Hyper-heuristic. Therefore, a completed initial candidate solution is required before the following optimization stage of the Hyper-heuristic approach can begin. In this research, Squeaky Wheel Optimization (SWO) has been selected to construct the initial feasible solution. The search space of SWO in this proposed approach is the solutions which fit the framework of the single-point perturbative Hyper-heuristic.

SWO is an adaptive iterative construction method. To use SWO to construct the initial solution, the Analyser, Constructor and Prioritizer must be implemented. In this research, weight is used as a numeric blame factor to represent the scheduling difficulty of each exam, and penalties are calculated to represent the violations by the corresponding exam of various soft constraints. In each iteration, the Constructor places one exam at a time based on the exam scheduling order given by the Prioritizer using a greedy algorithm. After the scheduling of the Constructor, the Analyser will increase the weight of the exam with the highest scheduling difficulty. The Prioritizer will then update the exam scheduling order based on the current weight of all the exams.

- **Constructor**: For the Constructor, the greedy algorithm will schedule each exam to the first available time and room combination while maintaining the feasibility of the incomplete solution and trying to minimize violations of the soft constraints. However, once an exam is unable to be scheduled in the current iteration, the exam will be left unscheduled and the Constructor will jump to the next exam. In each iteration, the Constructor will attempt to schedule all the exams.
- **Analyser**: Whenever an exam is successfully scheduled, the Analyser will calculate the penalty of the corresponding exam, which is defined by violations of various soft

constraints and add it to the existing weight store of this exam in the Prioritizer. If the Constructor failed to schedule the current exam, the current weight of this exam will be updated by adding an adequately large penalty, which is set to 2880 as recommended in (Hamilton-Bryce et al. 2014) in this research, to its existing weight store in the Prioritizer. The weights of all the exams held in the Prioritizer evolve over the duration of the entire construction process.

- **Prioritizer**: Priority sequence is the core of an SWO approach. The design of the calculation of the weight, as well as the update of the priority sequence, is crucial in the construction phase. In the literature, the means to determine the initial priority sequence varies. In this research, the initial priority sequence is generated by setting the weight to calculate result of each exam based on exam size and the number of conflicts. In each iteration, the Prioritizer will update the priority sequence of all the exams based on their weight. Specifically, exams with the highest weight value, which are the most difficult exams, will be moved to the beginning of the scheduling order to ensure they are scheduled first on the next iteration by the Constructor.

4.3.1 Parameters and Variants

Parameters

- Penalty: used to represent punishment of violations to the soft constraints.
- Maximum Penalty: will be assigned to an exam that has failed to be scheduled. This is set to 2880.
- Weighting: weightings are used to help build the priority sequence of the Squeaky Wheel Optimization. Whenever an exam is scheduled, a weighting will be assigned

to this exam. If the scheduling of exam eI at timeslot tIi is successful, the weighting value will be based on the PenaltyAt(Ii, ti). Otherwise, the weighting value will be set to the Maximum Penalty.

- WeightedList: stores the accumulated weightings of all the exams.
- Unschedule: counts the exams left unscheduled.
- Eval: represents the quality of the current complete/partial solution.
- Bestival: represents the quality of the current best complete/partial solution.
- SuitableTimeslots: a list built for each exam which includes all the available timeslots for this exam.

Mutators

- penalty(ei, ti): calculate the violations of exam Hi to all the soft constraints when scheduled at timeslot ti
- ScheduleAt(ei, ti): schedule exam ei at timeslot ti
- Sort(): resort the weightedlist to update the priority sequence
- CanSchedule(ei, ti): check whether schedule exam ei at timeslot ti is feasible; if not, check the violations to the hard constraints
- Unschedule(ei): Cancel the schedule of exam ei
- Evaluate(): calculate the penalty value of the given solution

4.3.2 Pseudocode of SWO Construction Algorithm

Table 9:Algorithm – Squeaky Wheel

Squeaky Wheel
Read in the problem file into memory and build conflict and suitability matrices;
Calculate an initial weighting based on pre-defined criteria;
WeightedList is used to store the scheduling order of all the exams
1. while stopping criteria not met **do**
2. **foreach** exam ei in weightedList **do**
3. **for** all suitable timeslots ti of ei **do**
4. **if** CanSchedule(ei, ti) **then**
5. best penalty and store best (bestti); //minimize violation to soft constraints
6. **endif**
7. **endfor**
8. **if** multipleBest found **then**
9. Schedule(ei, randomBestti) and store associated weighting in weightedList;
10. **endif**
11. **else if** bestTi found **then**
12. Schedule (ei, bestti) and store associated weighting in weightedList;
13. **endif**
14. **else**
15. Leave exam unscheduled and add large weighting in weightedList;
16. **end**
17. **endfor**
18. Sort(weightedList); //leave hard-to-schedule exam to later
19. endwhile

In table 9, the pseudocode of the SWO used in the proposed RL-SA-HH is given. In the pseudocode, bestti refers to the global best timetable solution while randomBestti refer to a random one from a set of current best timetable solutions.

4.4 Hyper-Heuristic Optimization

The proposed Hyper-heuristic framework is a perturbative, single-point heuristic selection Hyper-heuristic. The initial solution is constructed using the construction methodology described above. As the two main components of the heuristic selection Hyper-heuristic, the selection method and the move acceptance are the main focus of this research. In the

Hyper-heuristic described in this chapter, selection methods are inspired by the mainstream Reinforcement Learning algorithms in the literature. The move acceptance, which here is implemented using the Simulated Annealing algorithm, is designed to be a non-deterministic method. The research on various approaches used to escape the local optima with SA is also included.

Nevertheless, the research on the development of various low-level heuristics has also informed the implementation of the proposed Hyper-heuristic. The proposed framework can be illustrated in the Figure 9 below.

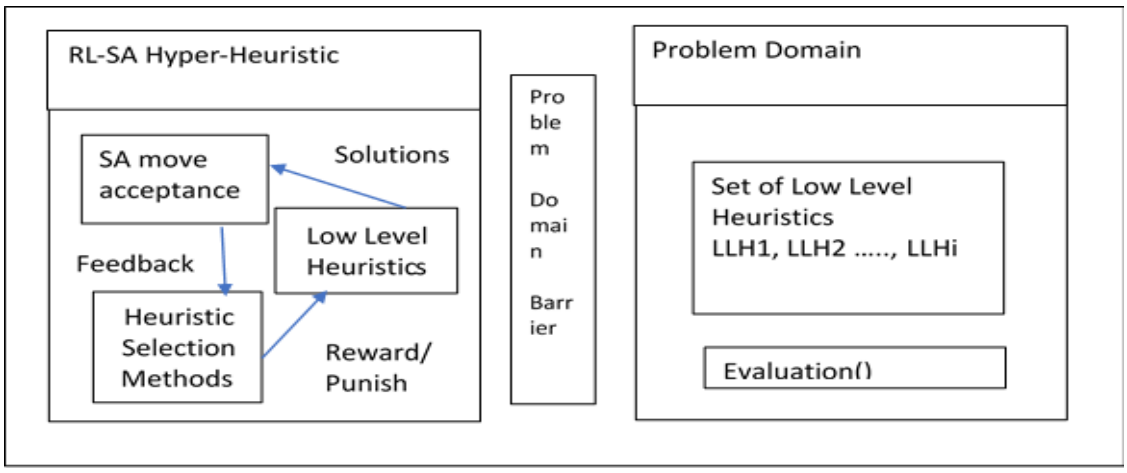

Figure 9:RL-SA Hyper-Heuristic Framework

The heuristic selection scheme consists of two main components: the selection method itself as well as the reward/punish function used to help the selection method to learn. The online learning heuristic selection Hyper-heuristic method has already introduced Reinforcement Learning trial and error in the research area in the literature (Özcan et al., 2012). However, the main popular algorithms of Reinforcement Learning have not yet been combined into the research area on the Hyper-heuristic. The Reinforcement Learning

techniques are selected based on their popularity. This is because the popular RL techniques are well applied in various optimisation problems in the literature which proves these techniques' potential.

In the literature, the performance of Reinforcement Learning in various optimization problems, as well as the wide application of Reinforcement Learning in the scheduling and resource allocating problems, has been proven to be discussed in section 2.4. In this proposed Hyper-heuristic, five different selection methods were developed and researched, together with four different reward/punishment schemes that are popular in the Reinforcement Learning area.

The selection method part can be considered as the policy used in Reinforcement Learning approach during the learning process. The learning policy is used to guide the selection of the next action, while the selection method is responsible for deciding the low-level heuristic to be used in the next iteration. The design of the policy will control the trade between Exploration and Exploitation. Here, for the purpose of researching how to better maintain the balance between exploration and exploitation in the learning process, five different selection methods are proposed and implemented based on four different popular learning policies of Reinforcement Learning algorithms.

4.4.1 Parameters and Mutators

Parameters

- ε: used in ε-greedy selection methods (section 4.4.5) to control the ratio between random selection (section 4.4.2) and greedy selection (section 4.4.4).

- τ: used in the Softmax selection method (section 4.4.7) to represent the temperature factor. The temperature is added into Softmax to change the probability distribution.

Variables

- Index: refers to the selected low-level heuristic index.

Functions

- llhMaxUtilityIndex(): A method designed to identify the low-level heuristic with the best performance.
- randomDuplicateSelection(): A method designed to locate the heuristics with the same utility value as the best low-level heuristics.

4.4.2 Random Heuristic Selection Approach

In this research, the design of the heuristic selection methods is entirely inspired by the Reinforcement Learning algorithms. For comparative purposes, a random heuristic selection method with no guidance or learning involved was implemented and tested.

Table 10:Algorithm – Random Selection

Random Selection

input: iteration: current iteration times; llhList: ArrayList of all the low-level heuristics
1. int index;//index of low-level heuristic
2. Random randInt = new Random();
3. index= randInt.Next(0, llhList.Count);
4. **return** index.

4.4.3 Equal Sequence: Exploitation Only

Exploitation only methods, also known as Maximize Explore theory, is actually a method that does not use the information gathered during the learning process to optimize the learning policy itself. This method provides each action with an equal opportunity during the exploitation learning process. Inspired by this theory, in the design of the heuristic selection methods, an Equal Sequence heuristic selection method is implemented by selecting each low-level heuristic with an equal number of times in sequence to ensure each is applied in equal measure. Specifically, the Equal Sequence selection method will select the low-level heuristics in sequence repeatedly to ensure that all low-level heuristics have an equal chance of being selected during the search process.

Table 11: Algorithm – Equal Sequence

Equal Sequence
input: iteration: current iteration times; llhList: ArrayList of all the low-level heuristics
1. int index;//index of low-level heuristic
2. int num= llhList.Count;
3. index=iteration/num;
4. **return** index;

4.4.4 Greedy-Exploration

The greedy policy is one of the most used policies in Reinforcement Learning. This policy always selects the best action for the next move. The advantage of this policy is that it manages to lead the learning towards the best result and makes the learning process converge quickly. However, if any move is overlook, it is easy to get trapped with local optima using this policy. It only takes the current immediate reward into consideration, which could miss delayed information during learning.

To implement this as a selection method, the actions are the low-level heuristics. Additionally, in each iteration, only the heuristics with the highest utility value will be selected to search the exam timetabling problem domain. Also, during each selection process, all of the low-level heuristics will be crossed to confirm whether there are not more low-level heuristics with the same utility value as the one selected. If so, one of the low-level heuristics with the same utility will be selected randomly. That is to say, at the beginning, when all the low-level heuristics are given the same initial utility value, the selection between them is random.

Table 12:Algorithm – Greedy

Greedy Selection
input: iteration: current iteration times;llhList: ArrayList of all the low-level heuristics
1. int index;//index of low-level heuristic
2. index = llhMaxUtilityIndex(llhList);
3. index = randomDuplicateSelectionSum(index, llhList);
4. //if there are more than one low-level heuristics with the same utility value
5. //select one from randomly
6. **return** index;

4.4.5 ε-Greedy

Due to the disadvantage of the greedy policy discussed in section 4.4.4, a probability parameter ε is introduced to control the balance between exploitation and exploration. The process of this algorithm is, in each step, with probability ε, the exploitation will be executed (select a random action); with probability 1-ε, the exploration will execute (select an action with the highest average reward, when more than one, select one from them randomly). Regarding the tuning of the probability ε, the higher the ε value the greater the exploitation and the smaller the ε value the greater the exploration. In the literature, the recommended setting value of the parameter ε normally is 0.01 to 0.1.

To introduce the ε-greedy policy into the heuristic selection process, an ε-greedy selection heuristic method is designed and implemented. A parameter ε is initialized at first with a reasonable value (between 0.01 and 0.1) to control the ratio of random heuristic selection and greedy heuristic selection. In each iteration, a random number is generated between 0 and 1. If that number is smaller than ε, we select a low-level heuristic randomly and update its utility value; or else, if it is equal to or bigger than ε, we select the low-level heuristic with the highest utility value (or a random one from the low-level heuristics with the highest equal utility value) and update its utility value.

As with greedy selection, the selection between low-level heuristics with the same utility value is random. Therefore, before all low-level heuristic have been selected at least once, random selection is employed. To determine the most suitable value for the greedy probability ε, experimental tests were performed with ε set to 0.1 as well as ε set to 0.01.

Table 13:Algorithm – ε-Greedy

ε-greedy Selection
input: iteration: current iteration times;llhList: ArrayList of all the low-level heuristics
1. int index;//index of low-level heuristic
2. int ε*=0.1;//or set* ε*=0.01;*
3. Random randDouble = new Random();
4. **if** (randDouble.NextDouble() < ε|| !checkSelected(llhList)){
5. //under ε probability, do random select
6. Random randInt = new Random();
7. //if there are more than one low-level heuristics with the same utility value
8. //select one from randomly}
9. **else** index = llhMaxUtilityIndex(llhList);
10. **return** index;

4.4.6 ε-Decay-Greedy

Although the ε-greedy policy manages to bring more exploration to the greedy policy using probability parameter ε, less exploration is required than at the beginning as the algorithm

learning is ongoing. This is because, as the algorithm learning increases, overexploitation becomes a burden of the computational cost. Therefore, it is claimed in the literature that the value of ε dynamic would be more suitable for the learning process. Specifically, the value of the parameter ε reduces as the learning continues. This is called ε –decay-greedy policy, and the decay rate of the parameter normally uses the formula $\varepsilon = 1/\sqrt{t}$ (where *t* is the current total number of iterations). Thus, the value of ε will be reduced gradually, and the possibility of the algorithm exploitation will decrease with the algorithm learning.

The implementation of the ε–decay-greedy heuristic selection is similar to the previous method, only the value of the greedy probability ε is dynamic. As the optimization process continues, the value of ε reduces to reduce the percentage of random selection. Thus, this in line with theory that the random low-level heuristic becomes less necessary as the Hyper-heuristic algorithm learns.

Table 14:Algorithm – ε-Decay-Greedy

ε-decay-greedy Selection

input: iteration: current iteration times;llhList: ArrayList of all the low-level heuristics

```
1.  int index;//index of low-level heuristic
2.  double edecay = 1 / (System.Math.Sqrt(iteration));
3.  Random randDouble = new Random();
4.  if (randDouble.NextDouble() < edecay || !checkSelected(llhList)){
5.      //under edecay probability, do random select
6.      Random randInt = new Random();
7.      index = randInt.Next(0, llhList.Count);}
8.  else{
9.      index = llhMaxUtilityIndex(llhList);
10.     index = randomDuplicateSelectionSum(index, llhList);}
11.     //if there are more than one low-level heuristics with the same utility value
12.    //select one from randomly
13. return index;
```

4.4.7 Softmax

In this learning policy, each action will be allocated a probability (based on the Boltzmann distribution). The selection among these actions will be made according to their actions, and the higher the action probability, the bigger the chance it could be selected. Specifically, in each selection, the method is to choose action k based on probability as follows:

$$P(k) = \frac{e^{-\frac{Q(k)}{\tau}}}{\sum_{i=1}^{K} e^{\frac{Q(i)}{\tau}}}$$

In this formula, Q(k) is the average reward, τ is a temperature parameter which is used to control the distribution, and e is the Euler's number, e=2.718.

To implement the Softmax policy as a heuristic selection method, the utility values of all low-level heuristics are recorded and corresponding Softmax probability values are calculated accordingly. It should be noted that the sum of all low-level heuristics probabilities is 1. In each trial, a random double value p is generated between 0 to 1 and compared to the probability of each heuristic. Subsequently, the low-level heuristic with the smallest probability of those low-level heuristics whose probabilities are bigger than random value p will be selected. After each selection, the utility values of all the low-level heuristics are updated.

As for the tuning of temperature parameter τ, the smaller τ is the higher the chance that the low-level heuristic with highest utility value will be chosen. Therefore, experiments with different temperature settings are performed to see the results. The value settings included τ =0.01, τ =0.5, τ =1, τ =10.

This Softmax policy-based heuristic selection begins with the calculation of the Softmax probabilities of all the low-level heuristics based on their utility values. The Softmax probabilities which guide the heuristic selection process are linked to the utility value. Therefore, in the beginning, all the low-level heuristics are selected at least once, and the selection is random as the Softmax probabilities will be the same. It should be noted that Softmax probabilities do not use accumulated utility update methods but rather the average utility values.

Table 15:Algorithm: Softmax Selection

Softmax Selection

input: iteration: current iteration times;llhList: ArrayList of all the low-level heuristics

```
1.  int index;//index of low-level heuristic
2.  double st=0.01;//set the τ =0.01 or set  τ =0.5 or to τ =1 or τ =10
3.  double currentLowestProbability = -1;
4.  double maxSoftmaxProbability = -1;
5.  double sumSoftmax=0; //use to calculate the Softmax probability of all the heuristics
6.  double Softmax[llhList.Count];
7.  //create array Softmax[] to hold the Softmax probability for all the heuristics;
8.  for(int i=0;i<llhListCount;i++)
9.      sumSoftmax += System.Math.Exp(llhList[i].AveUtility / st);
10. for(int i=0;i<llhListCount;i++)
11.     SoftmaxProbability[i] = (System.Math.Exp(llhList[i].AveUtility / st)) / sumSoftmax;
12. for (int i = 0; i < llhList.Count; i++)
13.     if (selectionProbability >= SoftmaxProbability[i] && currentLowestProbability <
    SoftmaxProbability[i])
14.         index = i; currentLowestProbability = SoftmaxProbability[i];
15. if (index == -1)
16.     return llhMinUtilityIndex(llhList);
17. index=randomDuplicateSelectionAve(index,llhList);
18. //if there are more than one low-level heuristic with the same utility value, select one randomly
19. return index;
```

4.5 Utility Update Methods

In Reinforcement Learning, the learning is based on the trial and error scheme, which learns through gathering information on the rewards/punish value of actions. However, learning

simply through immediate reward/punish actions is not adequate for long-term learning, as delayed information could be overlooked. Therefore, the utility value which is used to represent the actions within a continual learning process is crucial to the Reinforcement Learning. The function used to update the utility value is the utility function.

The Hyper-heuristic is also a trial-and-error learning process. The utility function is used to calculate the performance of each low-level heuristic, which is fundamental to the learning process of heuristic selection methods. Within the process of updating the utility values, a positive "reward" (increment) is given to a low-level heuristic if it was effective in generating a better solution, otherwise, a "punish" value is applied. For all utility update methods, the lower bound of the utility value is set to 0, the upper bound is set to 40 and the initial value is 20. These values were determined as a result of trial experimentation.

4.5.1 Parameters and Mutators

Parameters

- γ: the discounted factor used in the γ discounted incremental reward utility function. γ controls the influence of delayed reward information on the learning process.

Variables

- Utility: an attribute added to the low-level heuristics class representing the performance of low-level heuristics. Utility value can be either accumulated or average.

- rewardValue: the reward the low-level heuristic will be assigned to whenever an improving move is given. The rewardValue is set to 1 in this research.
- punishValue: the punishment/penalty value the low-level heuristic will be assigned to whenever a bad move is given. The punishValue is set to be -1 in this research.
- Upperbound: The Upperbound is set to 40 in this research.
- Lowerbound: The lowerbound is set to 0 in this research.
- Index: The index refers to the selected low-level heuristic index.

Functions

- updateUtility(): This method is designed to update the corresponding low-level heuristic with the given new utility value if the given value falls between the lowerbound value and the upperbound value.

4.5.2 Sum Utility

This is a simple, straightforward utility function that has been used in many Reinforcement Learning algorithms. This method simply accumulates the reward/punish value an action received during the learning process to update the utility value of that action. In the Hyper-heuristic, this utility function is implemented by updating the utility value of each low-level heuristic, which is done by adding the reward/punish value they were assigned during the whole learning process. In this method, the reward value is +1 and the punish value is -1.

Table 16:Algorithm – Sum Utility Method

Sum Utility Method
Input: bool flag: reward or punish, int index, llhList: heuristics ArrayList, SelectedTimes;
1. double newUtility=0;
2. **for** (int i = 0; i < llhList.Count; i++){
3. **if** (llhList[i].LlhId == index){
4. **if**(flag) newUtility= llhList[i].sumUtility+ rewardValue;
5. **else** newUtility= llhList[i].sumUtility+ punishValue;
6. updateUtilitySum (i, newUtility);}}

4.5.3 Monte Carlo

The Monte Carlo method (Doucet et al., 2001) learns directly from experiences, which in Reinforcement Learning is the performance of actions, and the estimations are simply based on calculating the Incremental Mean value of each action. The Monte Carlo method needs to track the previous performance of the target action to accurately estimate the Q value. In the Hyper-heuristic, the Monte Carlo method is implemented through calculating the average utility value of each low-level heuristic as a score of its perforce. As the learning continues, the Monte Carlo method is able to make the utility values of the low-level heuristics converge to stable accurate values.

In the literature, a paper managed to utilize stochastic Monte Carlo-based move acceptance methods to solve exam timetabling problems (Burke and Kendall et al., 2012). In this research, simple random (SR), greedy (GR), choice function (CF) and a learning scheme (L) are utilized as heuristic selection methods, and four low-level heuristics are designed for this algorithm (constraint-based). In the implementation of this method, the reward value is +1 and the punish value is -1. The sum utility value is gathered using the method described in the previous section and then divided by the iteration count to produce an average.

Table 17:Algorithm – Ave Utility Method

Ave Utility Method

```
Input: bool flag: reward or punish, int index, llhList: heuristics ArrayList, SelectedTimes;
1. double newUtility=0;
2. for (int i = 0; i < llhList.Count; i++){
3.     if (llhList[i].LlhId == index){
4.         if(flag) newUtility = ((SelectedTimes-1) * llhList[index].AveUtility + rewardValue) /
   SelectedTimes;
5.         else newUtility = ((SelectedTimes-1) * llhList[index].AveUtility
   +punishValue)/SelectedTimes;
6.         updateUtilityAve (i, newUtility); } }
```

4.5.4 γ Discounted Incremental Reward

This method is similar to the sum reward method mentioned above only with one more discount factor γ ($\gamma \epsilon (0,1)$). The reason for this is the belief that earlier rewards have a higher utility than later rewards. Also, it is a delay reward method that can help avoid getting caught by the local optimal. The function used to update the current utility value of action k is Q[k] = (q*t +r')/ (t+1).

In order to use this utility function in the Hyper-heuristic algorithm, a parameter γ is introduced as a discount factor when calculating the utility value of the low-level heuristics. Therefore, in iteration t+1, the current low-level heuristic utility value is updated using $\gamma^t r_{t+1}$. To set the parameter γ, in order to avoide over delay in the reward, it is recommended in the literature to set it as γ=0.9. This utility function is known as the Discount Sum Utility method. This method is similar to the sum utility update method. However, in this method, the positive reward and negative punish values are adaptive. The

positive reward value is set to be (+1) *(0.9^ (iteration times)), with the negative punish value set to be (-1) *(0.9^ (iteration times)).

Table 18:Algorithm – Discount Sum Utility Method

Discount Sum Utility Method
Input: bool flag: reward or punish, int index, llhList: heuristics arrylist, SelectedTimes, discountFactor=0.9.
1. double newUtility=0;
2. **for** (int i = 0; i < llhList.Count; i++){
3. **if** (llhList[i].LlhId == index){
4. **if**(flag) newUtility+= rewardValue * discountFactorR^selectedTimes;
5. **else** newUtility += punishValue* discountFactorR^selectedTimes;
6. updateUtilitySum (i, newUtility);}}

4.5.5 Temporal Difference

Temporal Difference (TD) is a combination of Dynamic Programming and Monte Carlo (MC). Both TD and MC use the experience to predict and solve the targeting problems. TD updates an immediate reward with an estimated value. The TD method updates the utility value of the current action *k* with the estimated reward in the next iteration with the incremental utility value in the experience. Therefore, there are two steps in the TD utility function. First, the utility value of action *k* in iteration *t* is updated based on experience using $Q[k] = \frac{q \times t + r}{t+1}$, where *r* is the immediate reward of action *k*. The utility value is then further updated using $Q[k] = \frac{q \times t + r'}{t+1}$, where r' is the estimated reward/punish value. The estimated reward/punish value r' is calculated based on the probability that action *k* might be selected in the next iteration.

Table 19:Algorithm – TD Sum Utility Method

TD Sum Utility Method

```
Input: bool flag: reward or punish, int index, llhList: heuristics ArrayList, SelectedTimes;
1.  double estimaterp = 0;
2.  double selectedProbability = 1.0 / NumLLH;
3.  double rewardProbability = 0.5;
4.  double newUtility = 0;
5.  for (int i = 0; i < llhList.Count; i++){
6.      if (llhList[i].LlhId == index) {
7.          if(flag) newUtility += rewardValue;
8.              else newUtility =punishValue;
9.          updateUtilitySum(index,newUtility);
10.         if (SelectedTimes != 0)
11.             rewardPrabability = (RewardedTimes + 1) / SelectedTimes;
12.         Random rand = new Random();
13.         if (rand.NextDouble() > rewardPrabability){
14.             estimaterp = selectedProbability * rewardValue;
15.             newUtility = llhList[i].sumUtility + estimaterp;}
16.          else{
17.             estimaterp = selectedProbability * punishValue;
18.             newUtility = llhList[i].sumUtility + estimaterp; }
19.       updateUtilitySum(index,newUtility); }}
```

In the Hyper-heuristic algorithm, for low-level heuristic k, the estimated utility value of the heuristic is again selected and is calculated and used to update the utility value of the heuristic. Firstly, the probability that the current low-level heuristic could be selected in the next iteration is calculated based on the utility value rank of this low-level heuristic (1.0 / (number of low-level heuristics)). Then, if selected, there is considered to be a 50 percent possibility that this low-level heuristic will perform positively. This estimated reward is then added to the current calculated sum utility to give final utility value of the low-level heuristic. The step size value has been experimentally tested between 1 and 5 to determine the most effective setting.

4.6 Simulated Annealing with Reheating Move Acceptance

In Hyper-heuristic algorithms, move acceptance is another crucial element that controls learning speed. In this research, the move acceptance is implemented with Simulated Annealing algorithms. The Simulated Annealing algorithms are designed with an annealing scheme as well as a reheating scheme to fulfill the role of move acceptance in the proposed Hyper-heuristic framework. The motivation for using Simulated Annealing as move acceptance arises from its ability to present a technique that is stochastic, easy coded and guaranteed to be good.

4.6.1 Annealing Scheme

The annealing scheme is the fundamental stage in the search process of the Simulated Annealing as it affects the overall performance of the Simulated Annealing (Abramson et al., 1999). The Simulated Annealing algorithm can be considered as stochastic hill climbing. It combines random research as well as a hill-climber algorithm using a temperature factor T.

During the searching process, the current solution is evaluated first. Then, if the evaluation result is good (the evaluation is descending), it will be accepted as the hill climber algorithm would. However, if the evaluation result is not better than the current solution (the evaluation is ascending), then it could be accepted within a probability P. Here, this probability P is used to control the ratio of hill-climber search to random search based on the Boltzmann distribution. Therefore, a worse move will only be accepted if it passes a random test, $\exp(-\Delta E/T)>\text{random}\ [0,1]$. The temperature factor T controls the

frequency of acceptance of worsening moves. For large T, this probability P is close to 1; with small T, the probability P is close to 0.

Within the proposed Hyper-heuristic framework, the annealing process is the process of searching for the solution with the lowest evaluation score. The lower the evaluated score, the less the violation of the constraints, hence the better the solution.

4.6.2 Cooling Schedule

During the Simulated Annealing search process, as the algorithm is annealing, the temperature is cooling to change the ratio of random search to optimal search. This is based on the theory that, as the result converges towards the best result, a less random search is necessary. The temperature T(k) is reduced gradually and the reduction is related to the number of iterations. There are two different types of cooling schedule. At each temperature, the search is allowed to proceed more than one step in the multiple cooling schedules.

In this research, the cooling schedule is single. The temperature is updated after each move. The cooling function used T(k+1) =T(k)*0.9 to decrease the temperature.

4.6.3 Reheating Scheme

The temperature T is used to control the percentage of random search in the Simulated Annealing. Therefore, when the search is trapped in the local optimal that for a period of time without finding an improvement move, the temperature T should be reheated to improve the chance that a random search will escape.

Here, this reheating scheme is introduced into the proposed Hyper-heuristic framework. Two steps in the reheating scheme are implemented in this research. First, a parameter reheat frequency is used to track and control the timing of the reheating process. That is to say, when no improvement move is found within the reheating frequency amount of iterations, the temperature is increased using T=T/0.85. Second, although the temperature is reheated to improve the random search, it could be insufficient if the temperature has converged too quickly. Therefore, a parameter maximum reheating time is used to trigger the second reheating stage, which involves resetting the current temperature to half of the initial temperature. In addition, the current best solution evaluation result fbest is also perturbed using fbest = (long)Math.Round(fbest * 1.1), this is out of the purpose that the solutions in the neighbourhood of the current best solution could be accepted.

4.6.4 SA and Hyper-Heuristic

In this research, a Simulated Annealing algorithm is implemented and embedded into the Hyper-heuristic as the move acceptance. In the literature, the introduction of the Simulated Annealing into the Hyper-heuristic has been investigated and implemented to solve various problems.

Bai and Kendall (2005) proposed a Simulated Annealing Based Hyper-Heuristic to realize Automated Planograms. Kalender et al. (2013) investigated how to use a greedy gradient-Simulated Annealing Hyper-heuristic to solve a curriculum-based course timetabling problem. Bai et al. (2007) implemented a Simulated Annealing Hyper-heuristic

methodology for flexible decision support, the results of which are promising and competitive.

Despite the many advantages of the Simulated Annealing algorithm, its disadvantages, for example, that it is time consuming, highly problem specific so that parameter tuning is difficult, and that it can produce misinterpreted results during the search process, cannot be overlooked. Combining it with this Reinforcement Learning based Hyper-heuristic framework proposed here will help to solve all these problems. As in the Hyper-heuristic, the search space of the Simulated Annealing is simply the moves provided by the low-level heuristics rather than the complicated exam timetabling problem domain. The problem domain is simplified and reduces the difficulty of the parameter tuning of Simulated Annealing.

4.6.5 Parameters and Mutators

Parameters

- T0: the initial temperature setting, which is set to 10000 in this research.
- reheatingFrequency: the reheat frequency setting. Experiments with frequencies of 1000 and 10000 are both performed.
- maxReheatingTimes: the maximum reheating limits, which is set to 5 in this research.

Variables

- Temp: this variable holds the current temperature value.
- Sbest: this is used to hold the current best solution.
- Fbest: this is used to record the quality evaluation result of the current best solution.

- Scurrent: this is used to hold the current temporary solution.
- Fcurrent: this is used to record the quality evaluation result of the current temporary solution.
- numNonImprovingReheat: this is used to track the number of times reheating has not given a better move.
- sinImp: this is used to track the number of iterations since the last improvement was found.
- bImprovementFound: this is a flag to remind the SA whether a better move has been found or not.

4.6.6 Process and Algorithm

Figure 10 below illustrates the Simulated Annealing working process in this research. The accompanying algorithm pseudocode is given in Table 20.

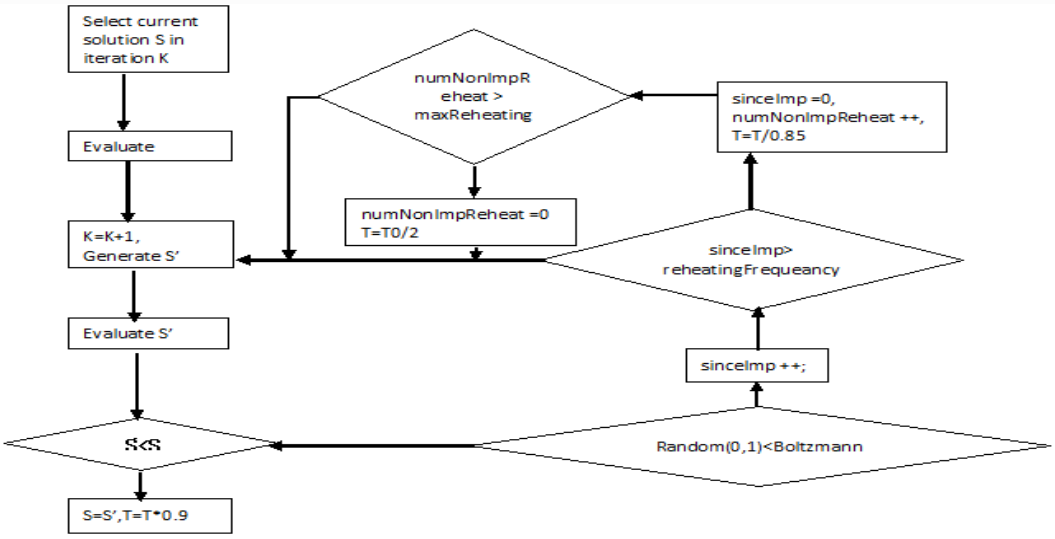

Figure 10:Simulated Annealing move acceptance

Table 20:Algorithm – RL-SA

RL-SA

Input –u: array holding utility value for each heuristic, totalTime, *T*0: initial temperature

```
1.  // initialisation
2.  Generate a random complete solution Scurrent ;
3.  Initialise utility values;
4.  fbest =fcurrent = f0 = quality( Scurrent );
5.  Sbest=Scurrent; //Sbest holds the best solution
6.  startTime = t = time();
7.  Temp=T0;sinceImp=0;
8.  numNonImprovingReheat=0;
9.  maxReheatTimes,=5;reheatFrequency=1000;
10.  bool bImprovementFound=false;
11.  // main loop executes until total running time allowed is exceeded
12.  while ( t < totalTime )
13. {//while running time still enough
14.   // heuristic selection
15.    i = selectHeuristic( u ); // select a heuristic using the utility values
16.     Stemp = applyHeuristic( i, Scurrent);
17.    //update temperare solution using selected low-level heuristic i
18.     ftemp = quality( Stemp );//update temperare evaluate value
19.    t = time() – startTime;//update remaining time
20.    // move acceptance
21.    Δ = ftemp − fcurrent;
22.    if (Δ < 0)
23.    {// improving move
24.       u[i] = reward( u[i] );Scurrent = Stemp;}}
25.    else u[i] = punish( u[i] ); // worsening move
26.    r ← random ∈ [0,1];
27.    if ( Δ < 0 ||r < P(Δ, Temp) )
28.    {// accept if non-worsening or with the Boltzmann probability of P
29.       if (Δ < 0) {
30.          Scurrent = Stemp;fcurrent=ftemp; bImprovementFound=true;}
31.       if (ftemp<fbest) Sbest = Stemp; else sinceImp++;}
32.    if (sinceImp==reheatFrequency)
33.    {//when reach non improvement limit
34.       sinceImp = 0; //reset since last improvement count to 0
35.       if(!bImprovementFound) numNonImprovingReheat++;
36.          else bImprovementFound=false;
37.       if(numNonImprovingReheat>maxReheatTimes)
38.       {//when reach max reheating limit
39.          numNonImprovingReheat = 0;
40.          Temp = (T0 - Temp) / 2.0;
41.          fbest = (long)Math.Round(fbest * 1.1);}
42.        else Temp = Temp / 0.85;//increase temperature}
43.     else Temp=Temp*0.9;// decrease temperature}
44. return Sbest;
```

4.7 Low-Level Heuristics

In this research, four different categories of low-level heuristics are proposed and implemented to improve the performance of the Hyper-heuristic framework. The first type of low-level heuristics consists of small perturbative moves that are either 1 operation or 2 operations. The second type low-level heuristics are relatively large perturbative moves that will move or swap a group of exams at one time. The exams are selected based on a random corresponding period, room or timeslot. The third type of low-level heuristics is shuffle heuristics with very large moves. This kind of low-level heuristics will shuffle all the exams in the period list, the room list and the timeslots. For example, for the period list, exams with period assignment p will be swapped with exams with period assignment p' where p' is random. The swap will continue until all the periods in the period list are visited. The last type of low-level heuristics is constraint-based. This kind of low-level heuristics will start the exam swap or move from the exams causing the most penalty. Here, the exams are selected based on the weightingList generated during the construction stage.

In this work, 19 low-level heuristics were implemented. Most of the low-level heuristics only perform simple moves for the purpose of ensuring that the Hyper-heuristic can recognize good moves and make an intelligent decision. Furthermore, this research aims to make the problem-domain knowledge heuristics easy to implement and the Hyper-heuristic more generalized. During the optimization process using the low-level heuristic, the moves suggested by the low-level heuristics must not violate any hard constraints to preserve the feasibility of the solution.

4.7.1 Parameters and Mutators

Parameters

- Exam: all the exams within the benchmarks.
- Room: all the rooms within the benchmarks.
- Periods: all the periods within the benchmarks; e.g. 9 am to 11 am is a period.
- Timeslots: comprised of a room and a period; e.g. a timeslot could be (room 201, period: 9:00-11:00).
- weightedList: a ranking of all the exams based on their scheduling difficulty generated by calculating their violations of both soft and hard constraints.
- softConstriantWeightedList: a ranking of all the exams based on their scheduling difficulty generated by calculating their violation of soft constraints only.
- Suitabletimeslot: also generated during the construction phase to help with the construction process, this is a list of all the timeslots into which the corresponding exam can be scheduled.

4.7.2 Small Perturbative Moves

The small perturbative series consisted of six, very simple move low-level heuristics. The simplest move in the optimization of the exam timetabling process is simply swapping and/or moving exams. This is the design concept of these low-level heuristics.

H1: random period assignment

In this low-level heuristic, a Random Exam in Period P1 with Room assignment R1 is selected. This exam is then assigned a different random Period P2 if feasible. During this process, only the period assignment is changed while the room assignment remains the same.

H2: random room assignment

In this low-level heuristic, a Random Exam in Period P1 with Room assignment R1 is selected. This exam is then assigned a different random room R2 if feasible. During this process, only the room assignment is changed while the period assignment remains the same.

H3: random timeslot assignment

In this low-level heuristic, a Random Exam in Period P1 with Room assignment R1 is selected. This exam is then assigned a different random Period P2 if feasible. This exam will also be assigned a different random Room R2 if feasible.

H4: random period swap

In this low-level heuristic, a Random Exam in Period P1 with Room assignment R1 is selected. Then, a Random Exam in Period P2 with Room assignment R1' is Selected. These two exams are swapped by assigning P2 to the first selected exam and P1 to the second selected exam if feasible. There will be no swapping of the rooms, thus the room allocation setting remains the same for the two selected exams.

H5: random room swap

In this low-level heuristic, a Random Exam in Room R1 with Period assignment P1 is selected. Then, a Random Exam in Room R2 with Period assignment P1' is selected. These two exams are swapped by assigning R2 to the first selected exam and R1 to the second selected exam if feasible. There will be no swapping of the periods, thus the period allocation setting remains the same for the two selected exams.

H6: random exam swap

In this low-level heuristic, a Random Exam with Room assignment R1 in Period P1 is selected. Another Random Exam Room assignment R2 in Period P2 is then selected. These two exams are swapped by assigning (P2, R2) to the first selected exam and (P1, R1) to the second selected exam if feasible. Each of these two exams is swapped to the other's Room (if both are feasible, e.g. if neither is exclusive).

4.7.3 Large Perturbative Moves

In this large perturbative series, the low-level heuristics will manage to make a group of moves or swaps to make a bigger change to the current solution to achieve better improvement and to speed up the search process.

H7: random group of exams swaps based on the period

This heuristic will select two Random Periods P1 and P2 and swap ALL exams between them. All the exams with period assignment P1 will be reassigned with period P2 as long

as it is feasible, while their room assignment remains the same. At the same time, all the exams with period assignment P2 will be reassigned with period P1 P2 as long as it is feasible, while their room assignment remains the same is feasible.

H8: random group of exams moves based on the period

This heuristic will select a Period P1 and move each exam assigned to that period to a new feasible period, one by one. That is to say, all the exams will be given a random new period assignment while their room assignment remains the same.

H7_2: random group of exams swaps based on the room

This heuristic will select two Random Rooms R1 and R2 and swap ALL exams between them. Specifically, all exams with room assignment R1 will be reassigned with Room R2 while their period assignment remains the same. Similarly, all exams with room assignment R2 will be reassigned with Room R1 while their period assignment remains the same.

H8_2: random group of exams moves based on the period

This heuristic will select one Room R1 and move each exam assigned to that room to a new feasible random room, one by one. The period assignments of all these rooms remain the same.

H7_3: random group of exams swaps based on timeslot

This heuristic will select two Random Timeslots T1 and T2 and swap ALL exams between them by assigning all the exams originally with timeslot assignment T1 with timeslot T2 and assigning all the exams originally with timeslot assignment T2 with timeslot T1.

H8_3: random group of exams moves based on the period

This heuristic will select one Timeslot T1 and move each exam assigned to that timeslot to a new feasible timeslot, one by one. The new timeslots are all randomly selected.

4.7.4 Very Large Moves

In this very large series, the suggested moves are all massive and for the purpose of shuffling the entire schedule whenever optimization is trapped.

H9: random group of exams shuffles based on the period

This heuristic is designed to shuffle all periods at random. This will use a loop to go through all the periods on the periods list. For each period, use low-level heuristic H4 to fulfill a random period swap between p1 on the periods list to a random p1'. Loop over all Periods i=P_1...P_K; H4 (i, Random (i+1, K)).

H9_2: random group of exams shuffles based on the room

This heuristic is designed to shuffle all rooms at random. This will use a loop to go through all the rooms on the rooms list. For each room, use low-level heuristic H5 to fulfill a

random period swap between r1 on the periods list to a random r1'. Loop over all Rooms i=R_1...R_K; H5 (i, Random (i+1, K)).

H9_3: random group of exams shuffles based on timeslot

This heuristic is designed to shuffle all timeslots at random. This will use a loop to go through all the timeslots on the timeslots list. For each timeslot, use low-level heuristic H6 to fulfill a random timeslot swap between t1 (r1, p1) on the periods list and a random t1' (r1', p1'). Loop over all Rooms L Loop over all Timeslots i=T_1...T_K; H6 (i, Random (i+1, K)).

4.7.5 Directed Moves

The directed series low-level heuristics are not just random move or swap; it is guided by using the weightedList as well as the softconstraintweightedList generated during the construction phase. These two lists rank all the exams based on their scheduling difficulty. This series of low-level heuristics are proposed for the purpose of balancing random moves with intelligent moves in the low-level heuristics stage to reduce randomness, which has the potential to be rather computational resource-consuming.

H10: ranked exams group move – consider both hard and soft constraint violations

This heuristic uses the weightedList, which considers all hard and soft constraint violations. It reallocates the top ten exams on the list to a random new timeslot. The new timeslot is selected from the suitable timeslot-List. The weighted-List is updated whenever a move is made.

H10_2: ranked exams group swap – consider both hard and soft constraint violations

This heuristic uses the weightedList, which considers all hard and soft constraint violations. From the top ten exams on the list, each exam with timeslot assignment t1 and a random exam with timeslot assignment t2 will be swapped. The weighted-List is updated whenever a move is made.

H11: ranked exams group move – consider only soft constraint violations

This heuristic will use the softconstriantWeightedList, which considers only soft constraint violations for the top ten exams on the list, reallocating them to a random new timeslot. The new timeslot is selected from the suitable timeslot-List. The softconstriantWeightedList is updated whenever a move is made.

H11_2: ranked exams group swap – consider only soft constraint violations

This heuristic uses the softconstriantWeightedList, which considers only soft constraint violations. From the top ten exams on the list, each exam with timeslot assignment t1 will be swapped with a random exam with timeslot assignment t2. The softconstriantWeightedList is updated whenever a move is made.

4.8 Evaluation Function

The evaluation function is used to evaluate the newly found solution by the low-level heuristics that represent the quality of the evaluated solution. At this optimization stage in the problem solving process, the evaluation function mainly involves evaluating the

violation of the soft constraints, as the optimization search is based on an initial feasible solution. During the search process, feasibility should be maintained.

4.8.1 Parameters

Parameters:

- exami: refers to an exam i.
- periodk: refers to a period k.
- timeslotj: refers to a timeslot j, a combination of period and room.

Functions:

- PenaltyAt(exami, timeslotj): this function calculate the penalty will be generated when schedule exam i at timeslot j.
- NumConflictsAt(exami, period): this function calculates the number of exams that are taken by the same students who are due to take exam i, and those exams are happen to be scheduled in period k.

4.8.2 The Evaluation Process

To further illustrate the evaluation process, dataset 1 from the ITC2007 benchmark is used as an example. The soft constraints setting in this dataset are listed in Table 21 as below. The detailed description of all the soft constraints of ITC2007 examination scheduling benchmark can be found in table 8.

Table 21:Soft Constraints in the ITC2007 benchmark

Soft Constraints	ID	Size	Penalty
S1.TWOINAROW			7
S2.TWOINADAY			5
S3.PERIODSPREAD		5 (Spread Size)	NUMofSTU
S4.NONMIXEDDURATIONS			10
S5.FRONTLOAD*	100(largest exams)	30(last periods)	5
S6.ROOM PENALTY	3	129	50
	4	60	50
S7.PERIOD PENALTY	4	210	70
	5	210	50
	20	210	50
	48	210	50

Breakdown

- Set the initial penalty as 0.
- For TWOINAROW, suppose there are two-period adjacent exams and both exams are taken by i students. The penalty here would be TWOINAROW=i*7.
- For TWOINADAY, suppose there are two non-period-adjacent exams but in the same day, and both exams are taken by j students. The penalty here will be TWOINADAY=j*5.
- For PERIODSPREAD, for an exam in a certain period, suppose that within the following five adjacent periods of this certain period there are k exams and h students are taking all k exams. The penalty here will be PERIODSPREAD =k*h.
- For NONMIXEDDURATIONS, for each timeslot, if there are n different sizes of exams, then the penalty will be NONMIXEDDURATIONS= n*10.
- For FRONTLOAD, if there are m exams from the 100 largest exams allocated in the 30 last periods, then the penalty will be FRONTLOAD=m*5.

- For ROOM PENALTY, if there are p number of exams allocated in a room with RoomID 3 and q number of exams allocated in a room with RoomID 4, then the penalty will be p*50+q*50.
- For PERIOD PENALTY, if there are r number of exams allocated in the period with PeriodID 4 and s number of exams allocated in the period with PeriodID 20, then the penalty will be r*70+s*50.

Therefore, the overall penalty = i*7+j*5+k*h+n*10+ m*5+(p*50+q*50) +(r*70+s*50).

In the optimization process within the proposed Hyper-heuristic framework, the solutions sent from the low-level heuristics are evaluated by the move acceptance method. The evaluation process for each received solution (timetable) is completed in four stages.

- **Stage 1**: For a received solution, get the penalty of each exam within this solution.
- **Stage 2**: For each exam, access its allocated timeslot and then calculate the penalty for scheduling this exam at this timeslot using PenaltyAt (exami, timeslotj).
- **Stage 3**: For each exam, using PenaltyAt() to sequentially calculate whether it violates six soft constraints with the help of function. Soft constraint penalties, including two in a row, two in a day and wider period spread, are all calculated with the help of NumConflictsAt (exami, periodk).

  For the calculation of soft constraint room penalty, period penalty and front-load penalty, simply compare the current evaluated exam room and period assignments with those which incur a penalty in PenaltyAt(exami, timeslotj).

- **Stage 4**: Calculate the mixed duration penalty separately once the penalty incurred by all the exams in relation to the six soft constraints has been calculated and add it to the current penalty. The mix duration penalty is calculated using a double loop.

4.9 Summary

In this chapter, the Reinforcement Learning – Simulated Annealing-based Hyper-heuristic is proposed and implemented. The proposed methodology is a single-point heuristic selection non-deterministic Hyper-heuristic. The framework is described and explained in detail, including the initial solution generation, heuristic selection, move acceptance and the design of the various low-level heuristics. The parameters that would improve the overall performance of the proposed framework during implementation are explained.

The research presented in this chapter includes a detailed analysis of different Hyper-heuristic algorithms. The Reinforcement Learning algorithms are introduced into the heuristics selection stage. The heuristics selection process is also a learning process for the Hyper-heuristic. The Hyper-heuristic manages to generate a better choice of the next selected low-level heuristics through the feedback gathered in the current iterations. The learning reflects on the estimation in the next iteration when selecting the low-level heuristic using the stored appearance (scores of the low-level heuristics).

The research motivation on the learning Hyper-heuristic is aims automate the design of heuristic methods to solve and search computational difficult problems. This study also explores the underlying strategic academic challenges to develop more generally applicable search techniques that can be brought into solving real-world problems.

In the next chapter, a cooperative Hyper-heuristic framework based on various cooperation schemes and agent designs is proposed and introduced in detail.

CHAPTER 5. COOPERATIVE HYPER-HEURISTIC

5.1 Introduction

In this chapter, the effect of using an agent-based cooperative Hyper-heuristic (COHH) framework to solve exam timetabling problems is investigated. Using an agent-based search enables the completed system to solve problems beyond the individual abilities of the single agents. The previous research result on Reinforcement Learning Simulated Annealing based Hyper-heuristic is used in the agent design portion of this research. The focus of the research is the cooperation scheme of the cooperative search. Different agent designs and cooperative search schemes are introduced and implemented to diversify the search process. This research aims to solve the problems of exam timetabling with its real-world benchmarks and complexity. The motivation of this research is to improve the generality and performance of the Hyper-heuristic search.

This chapter consists of four main sections: a discussion of the cooperative approach in exam timetabling, an introduction of the proposed COHH framework, two agent designs in the framework and a possible improvement approach for the synchronous search scheme.

5.2 Cooperative Hyper-Heuristic in Exam Timetabling

A cooperative search is an effective search that improves search efficiency by distributing the searching task into different sections to reduce the computational time cost. In the

literature, the cooperative search has been introduced into the scheduling research area and a promising result is given in (Ouelhadj and Petrovic, 2008). The combination of the cooperative parallel search with the Hyper-heuristic algorithms has also been performed and tested in solving flow shop problems.

5.2.1 Distributed Timetabling

A distributed framework has been introduced to solve the problem of exam timetabling Kaplansky and Kendall et al. in (2004). In the proposed model in this research, each agent is responsible for a campus. The distributive search starts from the local search within each agent for each campus, followed by negotiations with other agents for a feasible global solution. This approach benefits certain campus scheduling requirements that must be prioritised over others. In this paper, the proposed framework is implemented to solve the real-world exam timetabling problems from University Technology MARA in Malaysia. Dimopoulou and Miliotis (2004) studied the use of the computer network-based system to facilitate the construction of the course timetable for that university. Their proposed system uses an integer programming framework that allocates courses to suitable timeslots and rooms to construct a course timetable for each department. During the construction process, the data from each department is linked automatically to resolve conflicts generated by the distribution method. The proposed timetabling system has been tested with data provided by Athens University of Economics and Business.

Negotiation among scheduling agents for distributed timetabling problems was discussed in Kaplansky and Meisels (2004). Their research focused on the use of a Multi-Agent System (MAS) technology in solving exam timetabling problems. They used

constraint satisfaction problems techniques to design the algorithms in the scheduling agents. Four scheduling agents and a room agent are implemented to solve the course timetabling problems in this research. Each step in the negotiation process between these agents is discussed separately in this paper.

5.2.2 Cooperative Hyper-Heuristic

In Ouelhadj and Petrovic (2008), a cooperative distributed Hyper-heuristic framework for scheduling is presented to achieve a higher level of generality of metaheuristics. The paper investigates the influence of cooperative decision making on the heuristic selection part of Hyper-heuristic algorithms. This proposed cooperative distributed Hyper-heuristic framework is an agent-based system consisting of a high-level heuristic search agent as well as a series of low-level heuristic agents. The high-level heuristic agent controls the selection of the agents of the low-level heuristics. The agents of the low-level heuristics perform asynchronous searches on the same solution space, starting from the same initial solution and allocated different low-level heuristics. This proposed framework is then tested by solving the flow shop scheduling problems with promising results generated.

Ouelhadj and Petrovic (2010) introduced an agent-based cooperative Hyper-heuristic framework featured with a population of low-level heuristics agents. Their focus of their research is the investigation of cooperation between low-level heuristics within a Hyper-heuristic framework. Their research sought to compensate for the weaknesses of certain low-level heuristics with the strengths of other low-level heuristics. Both synchronous and asynchronous cooperative searches are implemented and tested in

(Ouelhadj, Petrovic 2010). Computational experiments using the proposed framework are given using a set of permutation flow shop benchmark instances.

5.3 Proposed COHH Framework

In this chapter, a general cooperative Hyper-heuristic framework is designed to solve exam timetabling problems. The proposed framework is implemented with various search mechanisms and agent settings. The motivation for using the cooperative search to solve exam timetabling problems is that it enables a wider search of the solution space within the same computational time. Also, the framework of the cooperative provides generality to the searching process. The search process is controlled by the mediator, but it can be altered by minor changes in the agents. Therefore, it is believed that the cooperative search combined with the Hyper-heuristic can improve the efficiency of the Hyper-heuristic methodology. The COHH is capable of providing a parallel search, which can reduce the overall time spent searching for the best solution.

The proposed Cooperative Hyper-heuristic Framework consists of three parts. Firstly, an initial solver is designed to construct the initial solution of the given exam timetabling problem. Secondly, a mediator is designed to faciliate cooperation and communication between each exam agent. Lastly, the exam agents are designed to optimize the initial solution in parallel using Hyper-heuristics with different parameter settings to implement various COHH approaches. The exam agents will preserve feasibility during the optimization process.

5.3.1 Cooperative Search

The cooperative Hyper-heuristic framework is an agent-based system consisting of a series of independent Hyper-heuristic exam agents which cooperate through a control agent mediator. The Cooperative mediator holds the global best solution, the priorities of the exam agents, and a pool of the best local solutions from the exam agents, as illustrated in Figure 11.

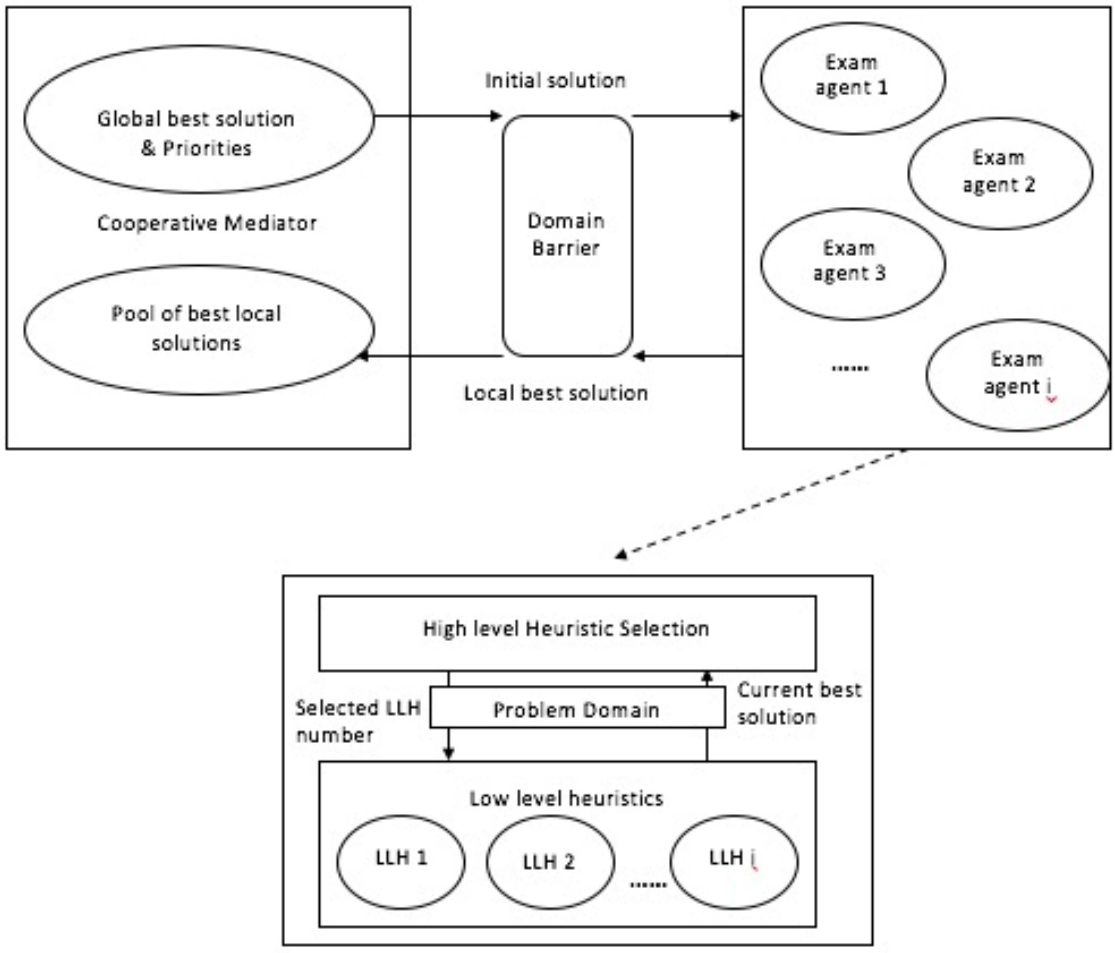

Figure 11:Cooperative HH Framework

The exam agents are responsible for local searches on the received complete initial solution sent by the cooperative mediator with the assigned Hyper-heuristic. They communicate with the mediator by exchanging their local solution to update the global solution for the next cycle. Here, two cooperative search mechanisms are proposed: synchronous and asynchronous. The details of these two cooperation processes are detailed in the following sections.

5.3.1.1 Synchronous search

In a synchronous search, the communication between the mediator and the agents occurs at the end of each cycle. The agents always search based on the same solution. At the beginning of each cycle, the mediator will send the global initial solution to each exam agent with the requests to perform optimization. One of the parameters that affects the synchronous search is the size of the cycle.

After receiving the request, the exam agents will start to search for a better move with its own algorithm. Whenever a better local solution is found, the search within this agent will terminate and the local best will be sent back to the mediator. Alternatively, if the maximum cycle length (iteration time limit) is reached, the search stops. The mediator holds a pool of local best solutions from all the agents. Once the mediator has received the local best solutions from all the agents it will generate, a global best solution accordingly.

The process is as follows:

- Mediator receives the initial solution from the initial solver which uses the Squeaky Wheel Optimization techniques introduced in Chapter 4. The received solution is used to update the global best solution.

- Mediator broadcasts the global best solution at the beginning of each cycle.
- For each received local best solution, the mediator will insert the received solution into the local solutions pool.
- At the end of the cycle, the best of the received local solutions is compared with the solution. If there are more than one best local solutions, the local solution forms with the highest priority agent. If it is better than the global best solution, then it is accepted and used to update the global best solution.
- If not, this solution could be accepted in accordance with acceptance criteria and be used to update the global best solution. Otherwise, the next best solution is checked with Simulated Annealing for acceptance or rejection. If no solution in the pool is selected, the global solution is sent back to the agents for a new search in the next cycle.
- The search of the cooperative Hyper-heuristic terminates whenever it reaches the maximum searching time. The solution pool will be reset by the mediator at the beginning of each cycle.

The pseudocode for the process is listed in Table 22 below. The mediator agent is referred to as MA, the exam agents as ETA, the global best solution as Sglobal and the local solution within the exam agents as Slocal.

Table 22:Algorithm – ETA in Synchronous Cooperative Search

ETA in synchronous cooperative search
1. **Receive** request search (MA,ETA, Sglobal,cycle-size)
2. Slocal=Sglobal;//Initialisation
3. int step=cycle-size;

```
4.  while(step>=0)
5.  {//local search
6.     local search for new Slocal';
7.     if(evaluate(Slocal')<evaluate(Slocal) or accepted with acceptance criteria)
8.        Slocal=Slocal';
9.     step--;}
10. Send (ETA, MA, Slocal);
```

In the following pseudocode, the initial solution is referred to as Sinitial and the best solution for all the local solutions is referred to as Slocalbest.

Table 23:Algorithm – MA in Synchronous Cooperative Search

MA in synchronous cooperative search

```
1.  Sglobal=Sinitial; arraylist solutionpool; array priorities;
2.  int cycle_size=10; int timelimit=300;
3.  int numofagents=4; /*or=6*/  bool accepted=false;
4.  int t=0; int cycle=0;
5.  while(t<timelimit){
6.     solutionpool.initial();
7.     Broadcast request search (MA,ETA, Sglobal,cycle-size);
8.     for(int i=0;i<numofagents;i++){
9.        Receive local best solutions (ETAi, MA, Slocali);
10.       solutionpool.insert(Slocali); }
11.    Slocalbest=best(solutionpool);
12.    int j=0;
13.    while((accepted==false) &&( j<solutionpool.size())){
14.       if (evaluate(Slocalbest)<evaluate(Sglobal)) {
15.          Sglobal=Slocalbest; accepted=true;}
16.       else if (accepted with acceptance criteria) {
17.          Sglobal=Slocalbest; accepted=true;}
18.       else {
19.          solutionpool.delete(Slocalbest); Slocalbest=best(solutionpool);}
20.      //go to the next best local solution.
21.       j++;}}
22. return Sglobal;
```

5.3.1.2 Asynchronous search

In an asynchronous search, the search between the mediator and the agents is exclusive. At the beginning of the overall search, all the agents are given the same initial solution. As soon as a better local solution is found, the agent will send it to the mediator and ask for feedback. However, it is possible the agent will not always be able to find a better solution within the given iteration limit. Therefore, the cycle length is important. If the agent fails to make improvements within a cycle (defined by number of iterations), it will request help from the mediator at the end of its search cycle.

The process is as follows:

- Mediator receives the initial solution form the initial solver using the Squeaky Wheel Optimization techniques introduced in Chapter 4. The received solution is used to update the global best solution.
- Mediator broadcasts the global best solution at the beginning of the first cycle.
- For each received local best solution, the mediator will insert the received solution into the local solutions pool.
- If the received local solution is better than the global solution, it is accepted and used to update the global solution. The sending agent will be requested to continue searching for the next cycle. If not, the solution could be accepted in accordance with acceptance criteria and used to update the global solution. Otherwise, the mediator will send the global solution back to the agent for a new search in the next cycle.
- If the agent fails to find a better solution within the cycle, it will request a new starting solution (the global solution) for the next cycle. Because the agent could get caught in the local optima, the new starting solution will be a random solution from

the local solutions pool (if the global solution is the same as the agent's current solution).

- The search of the cooperative Hyper-heauristic terminates whenever it reaches the maximum searching time. Whenever the solution pool is full, the worst local solution will be removed from the solution pool by the mediator to make room for the newly received local solution.

The pseudocode for the process is listed in Table 24 below. Note that the mediator agent is referred to as MA, the exam agents as ETA, the global best solution as Sglobal and the local solution within the exam agents as Slocal. Integer non-imp is used to measure the number of non-improvement iterations.

Table 24:Algorithm is exam agents

ETA in asynchronous cooperative search	
1.	**Reveive** request search (MA, ETA, Sglobal, cycle-size);
2.	Slocal=Sglobal;//Initialisation
3.	int step=cycle-size; int non-imp=0;
4.	bool improvement=false;
5.	while(step>=0) {
6.	local search for new Slocal' based on Slocal;
7.	if(evaluate(Slocal') <evaluate(Slocal) or within Boltzmann) {
8.	Slocal=Slocal'; **Send** (ETA, MA, Slocal);
9.	**Receive** feedback (MA, ETA, acceptance, Sglobal);
10.	if (feedback.acceptance == false) {
11.	**Request** help (ETA, MA, Slocal);
12.	**Receive** help (MA, ETA, new Sinitial); Slocal=Sinital;}}
13.	else {nonimp++;
14.	if (nonimp>=(cycle_size/2)) {
15.	**Request** help (ETA, MA, Slocal);
16.	**Receive** help (MA, ETA, new Sinitial); Slocal=Sinital; nonimp=0;}
17.	improvement =true;} step--;}
18.	**Send** (ETA,MA,Slocal, improvement);

Note that in the pseudocode, the acceptance is a bool variable used to reflect whether the received solution is accepted or not. Unlike the synchronous research,

immediate feedback is required by the ETA from the MA whenever a solution is sent. The new Sinital is used to help the ETAs to continue the local search whenever they get stuck with the search of the current Slocal solution.

Table 25:Algorithm – MA in Asynchronous Cooperative Search

MA pseudocode in asynchronous cooperative search

```
1.  Sglobal=Sinitial; arraylist solutionpool; array priorities;
2.  int cycle_size=10; int timelimit=300;int numofagents=4; /*or=6*/ T=10000; bool
    accepted=false;
3.  int t=0; int cycle=0; bool acceptance=false;
4.  Broadcast request search (MA,ETA, Sglobal,cycle-size);
5.  while(t<timelimit)
6.  {
7.     solutionpool.initial();
8.     //For each ETAi
9.     if(Receive (ETAi,MA,Slocal, improvement)==false )
10.       Send request search (MA,ETAi, Sglobal, cycle-size);
11.    else
12.    {
13.       Receive local best solutions (ETAi, MA, Slocali);
14.       if (evaluate(Slocali)<evaluate(Sglobal))
15.       {
16.          Sglobal=Slocali;
17.          if (solutionpool.size()==10) solutionpool.delete ( solutionpool.worst() );
18.          solutionpool.insert(Slocali); acceptance=true;
19.          Send (MA,ETAi, acceptance, Sglobal); }
20.       else if(accepted with acceptance criteria)
21.       {
22.          Sglobal=Slocali;
23.          if (solutionpool.size()==10) solutionpool.delete ( solutionpool.worst() );
24.          solutionpool.insert(Slocali); acceptance=true;
25.          Send (MA,ETAi, acceptance, Sglobal); }
26.        else //send new initial solution when rejected
27.       {
28.          Receive request-help (ETAi,MA, Slocal);
29.          if(Slocal!=Sglobal) Send help(MA,ETAi, Sglobal);
30.          else Send help(MA, ETAi, solutionpool.random() ) }}
31. //when receiving a request for new Sinitial
32. } return Sglobal;
```

5.3.1.3 Solution acceptance criteria

At each cycle, the mediator agent (MA) performs a search to gather the solutions from the exam agents (ETA). However, not all of the received solutions are the globally better

solution. Therefore, the acceptance criteria or the solution selection strategy is vital for the MA to decide whether to accept the currently received solutions or not. For the better solutions, whose evaluated value is lower than that of the current global value, acceptance is certain. However, the worse solution should also be taken into consideration for the next cycle. This is because the greedy acceptance method could easily cause the overall search to get stuck in the local optima. MA relies on the solution acceptance strategy to direct the overall search and send feedback to the ETAs.

5.3.2 Components

As described above, the COHH framework mainly consists of three parts: an initial solver, a mediator and exam agents. The details of the design and implementation of these components are given below.

5.3.2.1 Initial solver (constructor agent, CA)

The initial solver is responsible for the construction of a feasible initial solution. This initial solution is supposed to meet all the hard constraints to achieve feasibility. In the proposed COHH framework, the generation of the initial feasible solution uses the Squeaky Wheel Optimization (SWO) algorithm. The construction using SWO is an iterative adaptive process. When using the SWO construction process to solve timetabling problems, exams are scheduled using a greedy method according to sequence iteratively. The scheduling sequence is dynamic and best kept updated during the whole scheduling process. All exams are ranked based on their scheduling difficulty in the current condition.

Initially, exams are ranked based on a calculation result that considers the exams size as well as the number of conflicts. This ranking guides the schedule sequence during the whole construction process. In each iteration, after an attempt has been made to schedule all the exams, the ranking will be updated accordingly. Specifically, exams that have been scheduled in the current iteration will have a weighting, and the weightedList will be updated in terms of their current penalty. The penalty here refers to the violations of the soft constraints by the exam. A relatively large penalty value will be added to the current weighting value of exams that failed to be scheduled in the current iteration. This helps the initial solver to identify the difficult exams during the scheduling process. These difficult exams are the "Squeaky wheels" of the construction process and must be tackled with priority at the beginning of each iteration.

It should be noted that the weightedList here is also passed from the solver through the mediator to the exam agents in the second proposed COHH framework to assist the design of the low-level heuristics. In the COHH framework, the mediator sends requests for an initial solution with information about the target dataset and the construction time limit. The initial solver will send feedback with the construction result.

The pseudocode within the initial solver is given below to facilitate understanding of the construction process. It should be noted that CA refers to the initial solver/ constructor agent while MA refers to the mediator agent.

Table 26:Algorithm – Initial Solver

Initial Solver
1. if **Receive** request search(MA,CA, datasetNum, timelimit);
2. {
3. **Reading** datasets, generate exam ArrayList exams, Sinital=null;
4. bool constructed=false; int unsch=exams.size();
5. arraylist **weightedList**; int t=0;

```
6.        //t refers to Program.timer.Elapsed.TotalSeconds;
7.     bool scheduled=false; int unscheduledPenalty=2880;
8.     weightedList.initial();
9.        //generate the initial ordering to start the construction
10.    for(int i=0; i<weightedList.size(); i++)
11.       generate ArrayList timeslots;
12.       // generate the suitable timeslots for all exams
13.   // start construction
14.    while(t<timelimit && unshc>0){
15.       weightedList.Sort();
16.       for(int i=0; i<weightedList.size(); i++)
17.       {
18.          for (int j=0; j<exam[i].timeslots.size();j++)
19.          {
20.             if( CanSchedule(exam[i], timeslotj)
21.             {
22.                weightedList[i].update(penalty(exami,timeslotj));
23.                scheduled=true; Sinital.update(); unsch--;}
24.             if(scheduled==false) weightedList[i].update(unscheduledPenalty); }}
25.          if(unsch==0) constructed=true;
26.             Send (CA, MA, constructed ,Sinitial); }
27.    if Receive request search(MA,CA, datasetNum, timelimit, Sinital);
28.    Go to step 10;
```

The pseudocode within the mediator related to the initial solver is given in Table 27 below. It should be noted that CA refers to the initial solver, which is the constructor agent, while MA refers to the mediator agent. The time-limit given to the initial solver is sufficient for the initial construction through the tests. However, if the initial solver fails to construct the initial solution, the mediator will give the initial solver extra time for one more construction.

Table 27:Algorithm – MA-CA

```
MA-CA
1.  Send request search(MA,CA, datasetNum, timelimit);
2.  Receive result(CA,MA, constructed, Sinitial);
3.  if (constructed)
4.  Sglobal= Sinitial;
5.  else
6.  Send request search(MA,CA, datasetNum, timelimit);
```

5.3.2.2 Mediator

In the proposed COHH framework, the mediator agent is designed to be the agent that controls the overall cooperative search for the solutions to the exam timetabling problem. The mediator agent conducts the search using various exam agents instead of searching directly in the exam timetabling problem domain. The exam agents in this research are all designed to be Hyper-Heuristic. The whole search is coordinated by the mediator, including the start of the construction phase and the start of the optimization phase. The construction phase is completed by the initial solver and the constructor agent upon receiving the search request from the mediator. The optimization phase is allocated to the exam agents and starts receiving the search request from the mediator. The mediator is capable of diversifying the search process by changing the number of iterations, the time limit, and the initial solution that is sent to the exam agents.

Within the mediator, a solution pool is used to hold all the received local best solutions. This solution pool works differently in synchronous and asynchronous cooperative searches. In a synchronous search, the mediator only makes a decision until it receives information from all the exam agents. The solution pool is used to help the mediator make a decision as to when the mediator needs to generate the initial solution for the search of the exam agents in the next cycle. In an asynchronous search, the mediator communicates with the exam agents concurrently and the decision/feedback is given immediately. As the mediator is not going to communicate with all the exam agents at the end of each cycle, the exam agents need to communicate with the mediator whenever they get stuck during a search on the current local solution and need help. The mediator will then send a new initial solution to the requested exam agents. Although the global best

solution would be a good start for the requesting exam agents, sometimes they are simply stuck on the search on the global best solution. Therefore, a best solution is found from the solution pool and sent to the requested exam agents to diversify the local search.

Also, in the proposed framework, the exam agents are all isolated and only communicate with the mediator. The mediator holds all the transfer information and adjusts the overall search accordingly. In a synchronous cooperative search, the mediator needs to manage the conflicts between different exam agents. All the information received by the mediator will be used when it is time to negotiate between exam agents. The mediator also needs to consider the resource allocation when receiving more than one piece of information.

5.3.2.3 Resource allocation

In the proposed COHH framework, the local search within exam agents is designed to run concurrently. Therefore, the multi-thread method is used in the implementation of this research to realize the parallel search. Each agent is created as a thread and the mediator is the main thread that controls all the threads. Using threading in coding enables the programme to perform concurrent processing on multiple tasks. The multi-thread method is popular and promising in time-consuming tasks such as exam timetabling problem solving.

When using a multi-thread to solve problems, one of the most common issues is resource sharing during the search process. When a different thread is requesting overlapping sources for the local search, deadlocks or race conditions could occur and ruin the whole search. Therefore, in the proposed COHH, within the local search of each exam

agent, they are enabled to perform independent searches on the whole dataset. All the conflicts are handled in the mediator with the help of the solution pool. The exam agents will only communicate with each other through the mediator. This avoids the resource sharing problem during the local search process. As all the threads are performing searches concurrently, more than one message will be sent to the mediator waiting for a response. In a synchronous search, the response is given at the end of each cycle. Therefore, the order in which the message was received does not matter. In an asynchronous search, the exam agent whose messages are handled earlier gets to broadcast its own solution through the mediator. The sequence could verify the acceptance of the solutions sent out by the exam agents.

There are three main popular approaches in the literature to handling resource sharing problems. In this case, the resource is solved in the mediator. All messages are stored in a queue. The first is the First-in-First-Out approach, which does not consider any fairness problems. The first received messages get sent to the mediator and are processed first. The advantage of this approach is that it is computational cost friendly as there is no extra decision to make when handling messages.

The second approach is known as the Round-Robin, which aims at equal distribution when facing resource sharing problems. It can be implemented by providing each agent equal time in turn. However, as the searching speed varies between exam agents, this equality could slow down the search process when the quick exam agents have multiple messages waiting for responses, while slow or stuck exam agents have no message waiting. Both the first and the second approaches work provided no priority is taken into consideration.

The third approach allocates priority to different threads, so that the thread with higher priority can respond first when there is more than one message in the queue. In this research, the message queue is handled using the First-in-First-Out method as it is easy to implement and responds quickly to messages. All messages are handled in the order in which they were received.

5.3.2.4 Cooperation and negotiation

In a cooperative search, cooperation between the agents is based on the communication and interaction within these agents. The communication between agents refers to the messaging process within the cooperative framework which enables information exchange between agents. This communication enables the dynamic information to update within the whole system in order to achieve the global goal. The cooperation of both the sender and the receiver is required to complete an information exchange. All the agents are independent yet cooperative during the cooperative search.

Before sending a message, each agent needs to review its current search to generate (code) the appropriate message to push the search further or to ask for help. In addition, every agent, upon receiving the message, must deal with the message in three stages. First, the receiver needs to decode the received message to abstract the useful information. Secondly, based on an analysis of the abstracted information, the receiver needs to make further moves (in the proposed framework, these normally include accepting, rejecting or offering help). Finally, the receiver needs to respond to the received message by sending a message back with the decision that has been made.

As described above, in the framework proposed in this chapter, the communication is controlled by the mediator. Specifically, the exam agents do not communicate with each other directly but through the mediator. Also, the exam agents have all the necessary resources for their search of the local best solution. Therefore, the proposed framework avoids the conflicts during the local search by the exam agents. However, once all the exam agents have sent their local best solutions to the mediator, the mediator needs to pick one of these local best solutions for the acceptance criteria evaluation. Thus, the conflicts arise from here. The mediator needs to make decisions based on the received messages and negotiate with related exam agents if necessary.

In a synchronous cooperative search, the mediator sends response messages when all the messages from the exam agents have been received. The mediator needs to select the best solution from among all of the local best solutions in the solution pool. If there is more than one best local solution, the mediator needs to negotiate with the sending agents to decide which to accept. In this proposed framework, each exam agent is given a priority to help resolve this type of conflict. If more than one agent has the best local solution as well as the same priority value, the pre-accepted solution will be selected from them randomly.

In an asynchronous cooperative search, the exam agents communicate with the mediator as soon as they find a better local solution or when they reach the non-improving limit. Therefore, the conflicts between agents vary. Whenever receiving a solution, if its evaluated value is better (lower) or equal to the current global best solution, then it is accepted, or else accepted in accordance with the acceptance criteria. When one agent

sends the same solution that another agent has already sent, conflicts arise and are handled with the help of the evaluation function as well as the acceptance criteria.

5.3.2.5 Messages

The type of message sent between the exam agents and the mediator during the communication and negotiation process is given in Table 28 below. These messages can be categorized into five types: the request-search message, the inform-solution message, the feedback message, the help message and the request help message.

Table 28:Messages between MA and ETAs

Sender	Receiver	Message Types
MA	ETA	REQUEST-SEARCH
		FEEDBACK
		HELP
ETA	MA	INFORM-RESULT
		REQUEST-HELP

The details of each type of message are described in turn below.

Request-search: This type of message requests all the exam agents to apply their local Hyper-heuristic algorithms to conduct a local search for their local best solutions. It includes the following information:

- Sender: MA,
- Receiver: ETA
- Message Type: request-search
- Content: current global solution, next search cycle size
- Example: request search (MA, ETA, Sglobal, cycle-size)

Inform-solution: This type of message returns the search results from the exam agents to the mediator agents. It includes the following information:

- Sender: ETA
- Receiver: MA
- Message Type: information-solution
- Content: current local best solution, improvement flag value
- Example: result (ETA, MA, Slocal, improvement)

Feedback: This type of message returns the feedback from the mediator regarding whether to accept or reject the solution received from the exam agents. Also, a solution will be sent back to the corresponding exam agent for the next local search. It includes the following information:

- Sender: MA
- Receiver: ETA
- Message Type: feedback
- Content: acceptance flag value, current global best solution
- Example: feedback (MA, ETA, acceptance, Sglobal)

Request-help: This type of message is sent out whenever the exam agents are stuck in the search on their current solution and the search iteration numbers reach the non-improving limits. It sends out the local current solutions from the corresponding exam agents to diversify for the next local search. It includes the following information:

- Sender: ETA
- Receiver: MA
- Message Type: request-help
- Content: current local solution
- Example: request-help (ETA, MA, Slocal)

Help: This type of message returns a new initial solution in response to the exam agent's request for help. It includes the following information:

- Sender: MA
- Receiver: ETA
- Message Type: help
- Content: new initial solution (could either the current global solution or a random solution from the solution pool)
- Example: help (MA, ETA, new Sinitial)

In addition, there are negotiations between the mediator agent and the initial solver in relation to their common goal: the initial feasible solution. The messages sent and received between them include request search, feedback, and inform-result. The negotiation process is also simple compared to that between the MA and ETAs. The MA sends request-search messages to the CA, which refers to the initial solver and subsequently receives the inform-result message. Based on the received inform-result message, the MA either accepts the received initial solution or requests a further search for initial feasible solutions.

Table 29:Messages between Initial Solver(CA) and Mediator Agent(MA)

Sender	Receiver	Message Type
MA	CA	Request-search
CA	MA	Inform-result

The two types of messages are described in detail below:

Request-search: This type of message sends the target dataset number and the search time limit for the initial construction stage. This information includes:

- Sender: MA
- Receiver: CA
- Content: dataset number, time limit
- Example: request-search (MA, CA, datasetNum, timelimit)

Inform-result: This type of message returns the found solution as well as a flag value of whether the initial solution is constructed successfully or not. This information includes:

- Sender: CA
- Receiver: MA
- Content: constructed flag value, initial solution
- Example: inform-result (CA, MA, constructed, Sinitial)

5.3.2.6 Coordination

In the proposed cooperative Hyper-heuristic framework, coordination between the exam agents is also an important part. The cooperative search framework performs a search by

distributing the global search to different exam agents. The mediator coordinates the activities between different exam agents to ensure the overall cooperative search is towards the same global goal. Therefore, coordination is one of the vital aspects of the cooperative search framework. The coordination is defined as the management of dependencies between different agents by (Malone and Crowston, 1994). Coordination in the cooperative search framework can be realized through either direct or indirect interactions between agents. To manage the coordination in such a framework, the mediator agents need to take into consideration the states of all the agents.

5.3.2.7 Acceptance criteria in MA

In this research, the solution acceptance criteria are implemented using the Simulated Annealing (SA) algorithms. SA can be seen as a probabilistic hill-climber, which makes the search greedy most of the time, combined with a random search with Boltzmann Probability. Specifically, within the received solutions, the better solutions are received while the worse solutions will also be received with a probability exp(delta/T), where delta is the difference between the evaluated values of the received solution and the global best solution, while T is the temperature factor that simulates the annealing process. The acceptance of worse solutions within the SA searching process is controlled by the temperature factor T. The temperature T is reduced during the whole searching process. The temperature factor T is updated using T'=T*0.85 at the end of each cycle. The initial temperature is set to 10000. The temperature T is higher at the outset to provide the system with a higher proportion of random search. As the search proceeds, the random acceptance

of the worse solution will also reduce. The temperature T is used to balance the trade between the exploitation and exploration.

5.3.3 Two different COHH approaches

In this chapter, the general proposed COHH framework consists of three stages: the initial solution construction stage, the local solution search stage and the mediator solution selection stage. While the construction and solution selection stages stay the same even though two cooperative search schemes are designed and implemented, the agent search stage is designed into two different types. Therefore, in this research, through the design and implementation of the various components of exam agents, the proposed COHH frameworks can be divided into two types. For each agent setting, both synchronous and asynchronous searches are implemented and tested.

In both types of COHH framework, the exam agents conduct their local search through a Hyper-heuristic. The difference lies in the components of the Hyper-heuristic. In the first type of COHH, the agents are allocated with the same group of low-level heuristics but with a different objective value function. The different objective value function enables the corresponding exam agents to conduct local searches with a certain local goal which is controlled by the objective evaluation function. In the second type of COHH, the agents are assigned with a different set of low-level heuristics, but with the same objective value function. This design aims to provide the COHH framework with more possibilities by bringing adequate low-level heuristics into the cooperative search. In addition, each exam agent is given limited numbers of low-level heuristics in order to reduce the local Hyper-heuristic learning time.

The motivation for researching the cooperative search framework is that the cooperative search between the autonomous agents has enormous advantages for the general search of complex computational problems. The research on the cooperative search framework contributes to the development of complicated intelligent systems as well as hard resource allocation problems. The distribution approach enables the search program to solve the program more quickly and efficiently. However, the cooperation between different agents is always the focus of a cooperative search. In this research, it is proposed to implement the cooperative search by combining it with the Hyper-heuristic algorithms. The Hyper-heuristic algorithms are capable of solving complex problems and are feature with learning abilities. This suits the autonomous and independent requirements of the agents. The research described in this chapter is focused on how to coordinate the agents to achieve a common global goal. It also explores how to diversify the design of the agent to improve the efficiency of the proposed COHH framework.

5.4 Agent Design

As described above, in this research, two different agent designs are proposed and tested. Both these designs are complemented by allocating the exam agents with the same Hyper-heuristic framework. In the first COHH framework, the key component of the Hyper-heuristic, the objective evaluation functions vary in all the exam agents. By contrast, in the second COHH framework, in which the same objective evaluation function is given to all the exam agents, the low-level heuristics varies within each exam agent. Both these designs are presented and explained in detail below.

5.4.1 Soft Constraint-Directed Agent Design

In this type of COHH framework, the mediator holds the global objective evaluation function and is in control of whether or not to accept the solutions sent by the exam agents. At the same time, the exam agents perform the local agents with their own soft constraint-based local objective evaluation function.

The same three simple, low-level heuristics apply to all the exam agents. The objective evaluated function differs from one agent to another. Based on the ITC2007 dataset, six exam agents were designed, corresponding to six soft constraints. The Hyper-heuristic framework within each agent is basically the same apart from the objective value function.

5.4.1.1 Move acceptance in EA

The Hyper-heuristic framework implemented in the exam agents of the proposed COHH framework is a single-point, perturbative online learning Hyper-heuristic. The search process using this Hyper-heuristic can be considered to be an optimization process based on a feasible initial solution. During the search process, an acceptance criterion is used to decide whether to accept the new solution that has been found in the neighborhood of the current feasible best solution. This criterion is the move acceptance used in the Hyper-heuristic proposed in this COHH. The move acceptance must be able not only to accept the better solution in order to find the optimal solution but also to accept the worse solution for a reasonable probability of avoiding the trap of local optima.

The move acceptance used in the Hyper-heuristic in all exam agents is Simulated Annealing, which meets all the requirements mentioned above. The Simulated Annealing

algorithm searches for the best solution by accepting all the better and worse solutions within the Boltzmann solution. The searching process using Simulated Annealing algorithm includes a cooling scheme as well as a reheating scheme. In this research, the cooling schedule is single. The temperature is updated after each iteration. The cooling function uses T (t+1) =T (t)*0.9 to decrease the temperature. The reheating scheme used in this Hyper-heuristic framework includes increasing the temperature using T=T/0.85 when no improvement move is to be found within a reheating frequency move and resetting the current temperature to half of the initial temperature upon reaching the maximum reheating time.

All the better local solutions will be accepted and sent to the mediator, while all the non-improving local solutions will be accepted with Boltzmann probability. Specifically, if a proposed new solution from the low-level heuristic is better than the current local best solution, then it is accepted and used to update the current local best solution. If the proposed new solution from the low-level heuristic is worse than the current local best solution, then it is accepted with Boltzmann probability. The acceptance function is exp ((evaluate (new solution)-evaluate (current local best solution))/Temperature).

5.4.1.2 Heuristic selection

In the Hyper-heuristic searching process, the high-level heuristic does not perform a search directly in the problem domain of the exam timetabling. Instead, the higher-level heuristic performs searches in the domain of the selection of the low-level heuristics, then the selected low-level heuristic conducts searches in the domain of the exam timetabling problem. This process is the heuristic selection in the Hyper-heuristic. The heuristic

selection method is responsible for selecting the low-level heuristics and learning the features of the low-level heuristics is vital to the Hyper-heuristic algorithms.

To perform a heuristic selection process, the high-level heuristic must make selections based on the low-level heuristic searching performance. As mentioned above, each agent is assigned with three low-level heuristics. For each low-level heuristic, a utility value is used to track its searching performance. The heuristic selection process uses a ε-decay-greedy selection method. This method is a combination of random selections as well as greedy selection. In each iteration, within probability ε, the selection is random. Otherwise, only the low-level heuristic with the highest utility value (best performance so far) is selected. The factor ε controls the balance between greedy selection and random selection. As the Hyper-heuristic search continues, the value of factor $\boldsymbol{\varepsilon}$ decays to reduce the percentage of random selection. This is because when the Hyper-heuristic search iteration number arises, knowledge of the low-level heuristic performance grows. Factor ε is updated with $\varepsilon = 1/\sqrt{t}$ where t is the iteration number.

The utility values which are used to represent the low-level heuristic performance are updated with a discounted reward function. All the rewards or punishments received are added to the utility value of the corresponding low-level heuristic at the end of every iteration. The utility value method is called the γ discounted incremental reward utility method. Factor γ is set to 0.9 in this Hyper-heuristic.

The reward function here uses the r discounted reward function. For each low-level heuristic, their initial utility value is set to 20. In each iteration, if the selected low-level heuristic is able to find a better solution, a discounted reward is given (reward value is +1, discounted reward value is (+1) *(0.9^ (iteration times))) and added to its utility value to

update its performance. Otherwise, a discounted punishment is given (punish value is -1, discounted punishment value is (-1) *(0.9^ (iteration times))) and added to its utility value to update its performance. For each low-level heuristic, the utility value should be greater than 0 and less than 40. Therefore, if the utility value is less than 0 after being updated, the utility value stays at 0. Similarly, if the utility value is greater than 40 after being updated, the utility value stays at 40.

The utility method used here is sum utility value method. For each low-level heuristic, the initial utility value is set to 40. In each iteration, the reward/punish value is accumulated and updated as the new utility value.

5.4.1.3 Low-level heuristics

In this proposed COHH, each of the six exam agents is allocated with the same three low-level heuristics, which only make simple moves when selected. They are described as follows:

- **Move (Period)**: For details of this heuristic, refer to the description of H1 of the low-level heuristics in Chapter 4.
- **Move (Room)**: For details of this heuristic, refer to the description of H2 of the low-level heuristics in Chapter 4.
- **Swap (Timeslot)**: For details of this heuristic, refer to the description of H6 of the low-level heuristics in Chapter 4.

5.4.1.4 Differences between agents: evaluation functions

In this framework, six exam agents are designed to optimize the solution received from the mediator based on their own assigned soft constraints. This is achieved by giving all six exam agents different objective evaluation functions on their local solutions. Each is responsible for the optimization of soft constraints as outlined below:

- **Agent1**: Two exams in a row. Agent 1 is designed to optimize the current local solution aimed at reducing violations involving two exams in a row. This soft constraint is designed to avoid a situation such as a student sitting two exams one straight after another. In Agent 1, the local evaluation function will calculate the local newly found solution penalty for the violation of the two-exams-in a-row constraint. Specifically, for each exam, the periodID is abstracted. For the adjacent period within the same day, as in periodID+1, the number of exams allocated to this periodID+1 that also clash with the current exam is counted. The (counting result) * (Two in Row penalty value) is a current solutions penalty of the Two in Row constraint. The mixed duration constraint penalty is then calculated and added to the current penalty to generate the final evaluation result.
- **Agent2**: Two exams in a day. Agent 2 is designed to optimize the current local solution aimed at reducing violations of two exams in a day. This soft constraint is designed to avoid situations such as a student sitting more than one exam on the same day. In Agent 2, the local evaluation function will calculate the newly found local solution penalty for the violation of the two-exams-in a-day constraint. Specifically, for each exam, the periodID is abstracted. For the rest of the periods within the same day, as in [periodID+2 to period+i], the number of exams allocated

to these periods that also clash with the current exam is counted. The (counting result) * (Two in Day Penalty value) is a current solutions penalty of the Two-in-a-Day constraint. The mixed duration constraint penalty is then calculated and added to the current penalty to generate the final evaluation result.

- **Agent3**: the Period spread of examinations. Agent 3 is designed to optimize the current local solution aimed at reducing the violation to the period spread of examinations. This soft constraint is designed to allow an academic institution to 'spread' an individual's examinations over a specified number of periods. In Agent 3, the local newly found solution penalty for the violation of the exam period spread constraint. Specifically, for each exam, the periodID is abstracted. For the rest of the periods within the same day, as in [periodID+i to period+SpreadSize], the number of exams allocated to those periods that also clash with the current exam is counted. The counting result is the current solutions penalty of the Period Spread constraint. The mixed duration constraint penalty then is calculated and added to the current penalty to generate the final evaluation result.
- **Agent4**: Larger examinations appearing later in the timetable. In Agent 4, for each exam in the newly found solution, the number of exams which are among the largest exams (for example, the largest 100 exams) and also happen to be allocated in the last periods (for example, the last 30 periods) are counted. The (counting result) * (larger exam later period penalty) is the current penalty. The mixed duration constraint penalty is then calculated and added to the current penalty to generate the final evaluation result.

- **Agent5**: Period related soft constraints. In Agent 5, for each exam in the newly found solution, the allocated period penalty is checked. For those exams allocated in the period with a penalty, the penalty is calculated and accumulated to update the current penalty. The mixed duration constraint penalty is then calculated and added to the current penalty to generate the final evaluation result.
- **Agent6**: Room related soft constraints. In Agent 6, for each exam in the newly found solution, the allocated room penalty is checked. For those exams allocated in the room with a penalty, the penalty is calculated and accumulated to update the current penalty. The mixed duration constraint penalty is then calculated and added to the current penalty to generate the final evaluation result.

As the violation to the mixed duration of examinations reflects the balance of all the exams in the periods and rooms in the final allocation, it is designed as a common constraint which is optimized in all exam agents. This is also a cooperative process between exam agents.

5.4.1.5 Priority

In this proposed COHH framework, each agent is given a priority to help the mediator agent to solve conflicts. As discussed above, during a synchronous cooperative search, when more than one best local solution is sent to the mediator, the mediator will select one based on the priority of the sending agent.

Initially, the priority of each agent is set based on the penalty value. For example, for two exams in a row, the penalty is 7, while the penalty for two exams in a day is 5. In

this situation, having two exams in a row is allocated higher priority as it is believed to be more difficult to optimize compared to the other soft constraints. The priority is pre-set and doesn't change during the search process, although research should be conducted in future on the dynamic priority setting scheme.

5.4.2 Multiple Low-Level Heuristics Agent Design

In this type of COHH framework, the mediator holds the global best solution and is in control of whether to accept the solutions sent by the exam agents. At the same time, the exam agents perform as local agents with their own set of low-level heuristics. Unlike the previous COHH framework, the evaluation objective function in each exam agent is the same as the one used in the mediator agent. In this COHH framework, four exam agents were designed with four different low-level heuristics allocations. The other Hyper-heuristic settings within these exam agents are the same. This framework enables the possibility of bringing in various low-level heuristics for different target problems.

5.4.2.1 Move acceptance

In this proposed framework, the move acceptance is the same as that of the previous COHH framework. The move acceptance is a Simulated Annealing algorithm that completes the search with a cooling scheme as well as a reheating scheme. The cooling rate is 0.9 and is updated after each iteration. The reheating rate is 0.85 and is triggered whenever the non-improving iteration numbers reach the reheat limit. Moreover, the current temperature will be reset to half the initial temperature when the reheating time reaches the maximum.

5.4.2.2 Heuristic selection

In this proposed framework, the heuristic selection method is the same as that in the previous COHH framework. The heuristic selection method is a ε-decay-greedy selection method and the selection process is guided by the utility value of the low-level heuristics, which is updated using the γ discounted incremental reward utility method. The utility values represent low-level heuristic performance and are updated after the search in each iteration. Factor ε is updated with $\varepsilon = 1/\sqrt{t}$ where t is the iteration number. Factor γ is set to be 0.9 in this Hyper-heuristic.

The reward function is the update of the utility values. The utility values of all exam agents are initially set to 20 and the bond of the utility value is [0, 40]. The reward value is (+1) * 0.9^iterationNum and the punish value is (-1) * 0.9^iterationNum. The utility value method is the sum utility method. The utility value is updated by adding the newly received reward/punish value to the current utility value.

5.4.2.3 Evaluation functions

In this COHH framework, the evaluation function in all the exam agents is the same. The evaluation function considered the violation to all the seven soft constraints of the ITC2007 benchmarks (see section 4.8 for details).

5.4.2.4 Differences between agents: low-level heuristics

In this COHH framework, each exam agent has a different low-level heuristic setting. For Agents 1-4, the allocation low-level heuristics are Simple Moves low-level heuristics,

Group Moves low-level heuristics, Shuffle Moves low-level heuristics and Directed Moves low-level heuristics. The details of each of these are as follows:

- **Agent 1**: Simple moves. This includes low-level heuristics which only make 1-opt or 2-opt to the current solution for optimization. For details of the 6 low-level heuristics used in Agent 1, see the description of H1-H6 in Chapter 4.
- **Agent 2**: Group moves. This includes low-level heuristics that move or swap a group of exams that have been selected based on a certain room or a period. For details of the 6 low-level heuristics used in Agent 2, see H7, H7_2, H7_3, H8, H8_2 and H8_3 in Chapter 4.
- **Agent 3**: Shuffle Moves. This includes low-level heuristics which make shuffle moves or shuffle swaps of all the exam rooms or periods or timeslots. For details of the 3 low-level heuristics used in Agent 3, see H9, H9_2, and H9_3 in Chapter 4.
- **Agent 4:** Directed Moves. This will move or swap on the top 10 exams that have been selected based on their violation of all the constraints or just the soft constraints. For details of the 4 low-level heuristics used in Agent 4, see H10, H10_2, and H11, H11_2 in Chapter 4.

5.4.2.5 Priority

For the purpose of helping the mediator agent to negotiate with exam agents when facing conflicts, each agent is assigned a priority value. As described above in section 5.3.1, in a synchronous cooperative search, when there is more than one best local solution in the solution pool in the mediator, the mediator will make decisions according to the priority value of the corresponding agent.

As can be observed from the above description of the various low-level heuristics in all the exam agents, the move size of the exams suggested by different low-level heuristics set varies. Therefore, the initial priority setting is ascending based on the low-level heuristics move size. Specifically, for exam agents 1, 2, 3 and 4, the priority value is set as 1, 2, 3 and 4. The priority is pre-set and does not change during the search process. In future, however, the updating scheme possibly could be based on the exam agents' performance record after each cycle.

5.5 Improvement Trails based on Synchronous Search Scheme

Experimental testing of the asynchronous and synchronous schemes has shown that the performance of the synchronous scheme is poor. This is believed to be due to waiting time when trying to get solutions from all the Hyper-heuristic agents. In order to solve this problem, it is proposed to improve the synchronous communication scheme by receiving moves instead of solutions from the Hyper-heuristic agents.

This change was implemented on the 6-agent-based synchronous cooperative search. In each cycle, the agents get 10 iterations to perform local searches. Instead of sending the local best solutions to the mediator, the Hyper-heuristic sends an array list of all the accepted moves/swaps to the mediator.

In the 6-agent cooperative Hyper-heuristic framework, the Hyper-heuristic agents are all designed with simple low-level heuristics. Therefore, only three types of accepted moves/swaps will be sent to the mediator. These moves/swaps are: (exam1, room1, room2), (exam1, period1, period 2) and swap (exam 1, exam 2, timeslot1, timeslot2.) Having received all the moves/swaps from the Hyper-heuristic agents, the mediator needs

to handle the conflicts between them. Each received move/swap is accepted in sequence based on the priority value of the sending Hyper-heuristic. Violations of the hard constraints must be checked during whole acceptance process in the mediator. If one exam is moved to a new room/period twice, then the move/swap with higher priority is accepted and the lower one is rejected.

This proposed framework has been implemented and tested with datasets 7, 9, 11 and 12. However, the test result does not really improve the existing synchronous scheme. It is believed that the conflict handling process could be the reason for this. However, this is still a novel communication scheme and could use further search to discover more possibilities. The test result is given in Table 30 below.

Table 30:Multiple Moves, Synchronous 6-Agent-Based COHH Test Results

Datasets	**6-agent-based Syn search**	
	AVG	**BEST**
Exam 7	8112	5998
Exam 9	2107	1407
Exam 11	58200	45843
Exam 12	7452	6743

5.6 Summary

In this chapter, a complex agent-based cooperative Hyper-heuristic methodology is proposed to solve exam timetabling problems. The proposed cooperative Hyper-heuristic is described in detail, including various components of the framework as well as the search scheme, the cooperation method and the negotiation scheme. Conflicts in handling, one of

the main problems in the cooperative search framework, is also discussed in detail in this chapter.

The proposed cooperative Hyper-heuristic scheme was investigated by designing it as both a synchronous and an asynchronous search. The resource allocation method was set as First-In-First-Out in this framework. Two different settings were proposed for exam agents and implemented in order to observe the cooperation process within the Hyper-heuristic search process. The design of the first COHH framework aims to investigate cooperation between different agents when each agent has a different optimization purpose. The mediator agent manages to achieve the same global goal through coordination between different agents.

The second COHH framework sets each agent with the same optimization goal but with different local search techniques. This is achieved by allocating different low-level heuristics to the exam agents. Different local search techniques mean different search speeds, and this requires the mediator to coordinate between exam agents to complete the global search.

The research in this chapter introduces a novel cooperative Hyper-heuristic approach to solving complex ITC2007 exam timetabling problems. The experimental test results prove the capability of the proposed framework.

In the next chapter, the related experiments regarding the proposed frameworks in Chapters 4 and 5 are described and analysed in detail.

CHAPTER 6. RESULTS AND ANALYSIS

6.1 Introduction

This chapter will introduce the experiments that have been performed to test the RL-SA-HH framework and the COHH framework discussed in Chapters 4 and 5. As the proposed frameworks are rather complicated and composed of multiple components, adequate tests are implemented to find their best setting in order to generate promising results. The benchmark used for testing in this chapter is the ITC2007 benchmark, which was introduced and analyzed in Chapter 3. In this chapter, hypotheses related to the proposed frameworks are given first and verified subsequently.

The proposed RL-SA-HH framework and COHH framework are both implemented and tested on a PC with a 2.9 GHz Intel Core i7processor, 8GB RAM and macOS 10.12.05. The programming language used in both studies is in C# which targets the .NET Framework 4.5. As mentioned above, this test uses the ITC2007 benchmark. Therefore, for each run with the implemented program, the time limit is 270 seconds, which is generated using the benchmark tool from the ITC2007 website. During initial experimentation, it was found that allowing adaptive construction to execute for approximately 10% of the total execution time provided the best results. The proposed COHH approach is implemented as a multi-thread application in which all agents are implemented as threads. Communication between agents is completed using Microsoft Message Queuing (MSMQ).

This chapter consists of 3 main sections: (1) the research questions hypotheses related to the RL-SA-HH framework and the COHH framework, (2) the corresponding test designs and (3) the experimental results and analysis of the proposed tests.

6.2 Benchmark

The datasets used for all the experiments are from the ITC2007 benchmark, which is extracted from real-world exam timetabling problems, enabling the research in this study to be used in real-world problems. These datasets feature an adequate number of constraint settings.

The main features for each exam timetabling problem dataset of the ITC2007 benchmark are displayed in Table 6 in Chapter 2. The dataset with the smallest size is arguably either Exam 9 or Exam 12. The biggest exam datasets are Exams 3, 11 and 7. The features of Exams 3 and 11 are almost identical even though Exam 11 has a much smaller period size compared to Exam 3. For the test related to the variants setting, Exams 7, 9, 11 and 12 are selected as they include the largest and smallest data size as well as conflict density and provide adequate diversification for testing purposes.

The tests on the four datasets were used to guide the final design of the frameworks proposed in Chapters 4 and 5. For the overall evaluation of the two proposed frameworks for solving the exam timetabling problems, experiments were performed on all datasets from ITC2007 with the best variant settings. The best experimental results of the most promising settings for each framework were also compared to experimental results generated using other, previously published techniques used in solving the ITC2007 exam timetabling problems.

6.3 Hypothesis and Parameter Tuning Questions

In this section, for both the proposed RL-SA-HH framework and the proposed COHH framework, corresponding hypothesis and parameter tuning questions are proposed. The hypotheses are numbered after Capital character H while the questions are numbered after Capital character Q.

6.3.1 RL-SA-HH

The configuration of the variants matters, as the proposed framework is a complex system. Therefore, the experiments also have the purpose of configuring the best parameter selection in order to generate the best final result. These parameters are listed below. The exploration of the best configuration also contributes to the research on how to better combine the RL with HH.

Questions

The following questions are proposed during the development process of the RL-SA-HH framework. The questions are related to the design of 3 aspects: move acceptance, the choice of heuristic selection and utility update methods.

Move Acceptance-Simulated Annealing settings

Move acceptance is an important part of the Hyper-heuristic that controls the acceptance of the proposed moves received from the low-level heuristics. Specifically, in this RL-SA-

HH framework, move acceptance is implemented using the SA algorithm which is designed with a reheating scheme. The responsibility of the SA acceptance here is to lead the overall search towards the best final result without getting caught by the local optimal. The three main parts of the proposed SA algorithm in this HH are the reheating frequency, the reheating scheme and the temperature updating function (cooling scheme).

- Q1-1.1: What is the best cooling rate?
- Q1-1.2: What is the best reheating rate?
- Q1-1.3: What is the best reheating frequency?
- Q1-1.4: What is the value of the best reheating limit (5 vs 20)?
- Q1-1.5: Will resetting the low-level heuristics value make the RL-SA-HH perform better?

Heuristic selection method: Reinforcement Learning-based

The heuristic selection is the high level of the Hyper-heuristic method. This heuristic selection can be implemented either by using simple heuristics or in combination with machine learning methods. The role of the heuristic selection is to select the low-level heuristic for the search in the current iteration. In this proposed framework, heuristic selection methods which are inspired by the RL algorithms (in order to bring the learning feature to the heuristic selection stage) are implemented and tested. Experiments are required to identify the best variants settings for this selection method and then identify the best selection method.

- Q1-2.1: For the ε-greedy selection method, what is the best value for e?

- Q1-2.2: For the Softmax selection method, what is the best value for the temperature factor T?
- Q1-2.3: What is the best Heuristic selection method?

Utility update method: Reinforcement Learning-based

Another variant that could affect the heuristic selection process is the utility update method, which in this research is implemented based on the common reward function in the RL algorithms. During the heuristic selection process, decisions are made based on the performances (utility value) of the low-level heuristics. Therefore, the utility update method is vital to the heuristic selection stage and needs adequate experiments to determine the best one to use.

- Q1-3.1: For the discount sum utility method, what is the best value setting for the discount factor r?
- Q1-3.2: For the Temporal Difference utility method, what is the best value setting for the step size t?
- Q1-3.3: What is the best Utility Update Method?

In order to explore various neighbour domain spaces in the proposed RL-SA-HH framework, 19 low-level heuristics are implemented for the higher heuristic to select. However, the more choices there are, the longer the learning time required. Whether the RL-SA-HH performs better with more low-level heuristics or not must be verified.

Hypotheses

The RL-SA-HH framework was introduced and explained in detail in Chapter 4. The proposed Hyper-heuristic is mainly composed of heuristics selection and low-level heuristics. The heuristic selection consists of the move acceptance and utility update method. In Chapter 4, for each part of the proposed Hyper-heuristic, more than one design is proposed in order to determine the best choice for better final results. To analyze the influence of each section of the proposed RL-SA-HH framework, the following hypotheses are proposed for verification.

- H1-1: SA with reheating scheme performs better in solving exam timetabling problems.
- H1-2: Reheating limits help SA to perform better. Specifically, whenever the reheating limits are reached, reset the current temperature and increase the current best solution evaluation value.
- H1-3: RL-SA-HH is applicable for solving complex exam timetabling problem benchmarks.
- H1-4: RL-SA-HH is capable of generating competitive results in solving exam timetabling problems compared to other techniques used for the same purpose in previous studies.
- H1-5: RL-SA-HH is competitive in solving exam timetabling problems compared to other Hyper-heuristics used for the same purpose in previous studies.

6.3.2 COHH

A cooperative search is a complicated process which requires the communication and cooperation of all agents. Therefore, certain variants must be tested to get the best the result. In the proposed COHH framework, these variant configurations include the iteration limits for the exam agents in each cycle and the SA setting used in the mediator agent. Experiments with various settings are necessary and the experimental results are helpful in determining the best mediator agent setting. Also, experiments for the two different agent settings are helpful to define the best exam agents setting.

Questions

The following questions are proposed during the development process of the COHH framework. The questions relate to the design of the mediator agent (the SA algorithm as well as the cooperation scheme) and the exam agents.

SA in mediator agent

The mediator agent controls the cooperative search, and the instruction sent to the exam agents enables the search to continue. The mediator agent in the proposed COHH framework uses the SA algorithm to react to the messages it receives. Therefore, experiments to identify the best component variant setting of the SA algorithm implemented in the mediator are vital to the overall performance of the proposed COHH framework. The following question regards the variant settings of the SA algorithms.

- Q2-1.1 Is the geometric cooling/reheating scheme better than the enhanced geometric cooling/reheating scheme?

Other parameter settings in the mediator

In addition to the SA algorithm, there are numerous other settings that influence cooperation and communication between the agents in the mediator agent. The solution pool keeps all the local elite solutions sent by the exam agents in each cycle, while the cycle size determines the number of iterations in which the exam agents get to search for a better local elite solution. The following questions regard the variant settings of the SA algorithms above to aspect.

- Q2-2.1 What is the best solution pool size? Is it the number of exam agents or double the number of exam agents?
- Q2-2.2 What is the best cycle size for the exam agents?

Exam agents

The general design of the Hyper-heuristic framework within the exam agents is informed by the analysis of the experimental results from the research reported in Chapter 4. Therefore, the Hyper-heuristic implemented in either the 6-agent based-COHH framework or the 4-agent-based COHH framework has the same setting for the move acceptance, heuristic selection method and utility update method. In other words, there is no need for further experiments on these aspects of the exam agents.

However, experiments are required in the asynchronous cooperative search as the exam agents determine the timing of interventions by the mediator agents. Furthermore, more experiments are proposed to help identify variations in the performance of different exam agents when working alone.

- Q2-3.1 In the asynchronous cooperative search, what is the best value setting (number of non-improving iterations) for requiring intervention from the mediator agent in the exam agents?

Hypotheses

In Chapter 5, a COHH framework was introduced in detail. The proposed framework can be implemented with different cooperative schemes, and the various agent designs also provide different performance possibilities. The cooperation between the mediator agents and the exam agents can be divided into two types: synchronous and asynchronous. Two different designs for exam agents were introduced in Chapter 5. The basic HH framework of the exam agents in both designs is based on the analysis of RL-SA-HH related experimental results. However, the evaluation function used in the exam agents and the low-level heuristics assigned to the exam agents and the components differs for each exam agent. Here, several hypotheses are given accordingly.

- H2-1: The priority setting for the exam agents helps the mediator agent lead the overall search better.
- H2-2: The COHH framework with more exam agents performs better.

- H2-3: The asynchronous scheme is better at solving exam timetabling problems than the synchronous scheme in the proposed framework (Ouelhadj and Petrovic, 2008).
- H2-4: The proposed COHH framework, including the synchronous and asynchronous schemes, four agents and six agents design, is fully applicable for solving the ITC2007 exam timetabling problems.

- H2-5: By tuning the variants, the COHH framework can generate a completive result in solving exam timetabling problems compared to other, previously published methods.
- H2-6: The cooperative scheme is capable of improving the performance of a simple 2-stage Hyper-heuristic.

6.4 Experimental Design

In the following sections, based on the proposed questions and hypotheses for the RL-SA-HH and COHH frameworks, corresponding tests were designed as listed below. The tests are numbered after Capital character T.

6.4.1 RL-SA-HH

In this section, the experiments arising from the RL-SA-HH related questions and hypotheses which are proposed previously are described in detail as below. These experiments are designed and conducted with the aim of verifying these hypotheses and with the aim of answering these questions. For the purpose of finding solutions to the

problems relating to variant configuration, a series of experiments are designed as described below.

Questions related experiments

Based on the questions set out in Section 6.3.1 regarding the implementation of the RL-SA-HH framework, the following test designs are proposed.

Move Acceptance-Reheating frequency

Because it is believed it would be better to implement the reheating scheme for the SA algorithm used to move acceptance, the best value setting of the reheating frequency must be found through experimentation.

- T1-1.1: For the SA of the implemented program, results from experiments designed to solve ITC2007 exam timetabling problems, including setting the cooling rate as 0.5 and 0.85, should be collected for comparison in order to answer Q1-1.1.
- T1-1.2: For the reheating scheme in the SA of the implemented program, results from experiments designed to solve ITC2007 exam timetabling problems, including setting the reheating rate as 0.99 and 0.9, should be collected for comparison in order to answer Q1-1.2.
- T1-1.3: For the reheating scheme of the implemented program, results from experiments designed to solve ITC2007 exam timetabling problems, including setting the reheating frequency as 10000 and 1000, should be collected for comparison in order to answer Q1-1.3.

- T1-1.4: For the reheating scheme of the implemented program, results from experiments designed to solve ITC2007 exam timetabling problems, including setting the reheating limit as 5 and 20, should be collected for comparison in order to answer Q1-1.4.
- T1-1.5: For the reheating scheme of the implemented program, results from experiments designed to solve ITC2007 exam timetabling problems, including resetting and not resetting the utility value to original when reaching the reheating limit, should be collected for comparison in order to answer Q1-1.5.

Heuristic Selection Method: Reinforcement Learning-based

The heuristic selection is a crucial component of the Hyper-heuristic method. This heuristic selection can be implemented either by using simple heuristics or in combination with machine learning methods. The role of the heuristic selection is to select the low-level heuristics for the search in the current iteration. In this proposed framework, heuristic selection methods which are implemented are inspired by RL algorithms in order to bring the learning feature to the heuristic selection stage. Therefore, experiments are required to select the best selection method.

- T1-2.1: To determine the best value setting for the Softmax heuristic selection method, experiments with the ε set as 0.1 and 0.01 should be performed on solving the ITC2007 exam timetabling problems. The experimental results should be collected for comparison in order to answer Q1-2.1.
- T1-2.2: To determine the best value setting for the ε-greedy heuristic selection method, experiments with the temperature factor T set as 0.1 and 1 should be performed on

solving the ITC2007 exam timetabling problems. The experimental results should be collected for comparison in order to answer Q1-2.2.

- T1-2.3: To determine the best heuristic selection method, experiments using the equal sequence selection method, the greedy selection method, the ε-greedy selection method with the best ε setting, the ε-decay-greedy selection method and the Softmax selection method with best temperature setting should be performed on solving the ITC2007 exam timetabling problems. The experimental results should be collected for comparison in order to answer Q1-2.3.

Utility update method-Reinforcement Learning based

Another variant that could affect the heuristic selection process is the utility update method, which in this research is implemented based on the common reward function in the RL algorithms. During the heuristic selection process, decisions are made based on the performances (utility value) of the low-level heuristics. Therefore, the utility update method is vital to the heuristic selection stage and needs adequate experiments to determine the best one to use.

- T1-3.1: In order to identify the best discount factor setting for the discount sum utility update method, experiments with the temperature factor r set as 0.5 and 0.9 should be performed on solving the ITC2007 exam timetabling problems. The experimental results should be collected for comparison in order to answer Q1-3.1.
- T1-3.2: In order to identify the best step size setting for the Temporal Difference utility update method, experiments with the step size set as 1 and 5 should be performed on

solving the ITC2007 exam timetabling problems. The experimental results should be collected for comparison in order to answer Q1-3.2.

- T1-3.3: To determine the best Utility Update method, experiments using the Sum Utility Update Method, Average Utility Update Method, Discount Sum Utility Update Method with the best r setting and the Temporal Difference Utility Update Method with best Temperature step size setting should be performed on solving the ITC2007 exam timetabling problems. The experimental results should be collected for comparison in order to answer Q1-3.3.

Hypothesis-related experiment design

Based on the hypotheses proposed above, a series of experiments, described below, have been designed and await verification.

- T1-4.1: For the implementation of the proposed RL-SA-HH framework, the move acceptance SA should be implemented both with and without a reheating scheme to solve the ITC2007 exam timetabling problems. The experimental results from both implementations must be collected separately for comparison in order to verify H1-1. Specifically, for the implementation including the reheating scheme, the reheating will be triggered every 10000 iterations and the current temperature will be increased.
- T1-4.2: For the implementation of the proposed RL-SA-HH framework, the move acceptance SA should be implemented both with and without reheating limits to solve the ITC2007 exam timetabling problems. The experimental results of both implementations need to be collected for comparison in order to verify H1-2.

Specifically, for the implementation including reheating limits restrictions, the following action including increase the current temperature to halfway between the current temperature and the initial temperature, increasing the current best solution evaluation value by 110% will be taken.

- T1-4.3: The proposed RL-SA-HH framework should be implemented to an executable program and used to solve all the 12 datasets from the ITC2007 benchmark in order to verify H1-3.
- T1-4.4: The experimental results generated by RL-SA-HH need to be gathered and compared to other, previously published experimental results on the ITC2007 exam timetabling datasets in order to verify H1-4.
- T1-4.5: The experimental results generated by RL-SA-HH need to be collected and compared to other, previously published experimental results using Hyper-heuristic in order to verify H1-5.

6.4.2 COHH

Experiments that consider the questions and hypothesis proposed previous regarding the COHH framework are needed to support the research and analysis of the COHH framework working scheme with a case study on exam timetabling problems. For the variant set, queries regarding the SA setting in the mediator agent and related variant support in the communication and cooperation process, as well as the variables used in the exam agents, are proposed. In this section, tests are designed to identify the answers to these queries.

Questions related experiments

Based on the questions set out in Section 6.3.1 regarding the implementation of the COHH framework, the following test designs are proposed.

SA in mediator agent

The SA algorithm consists of the cooling scheme, the reheating scheme and the termination criteria. These aspects have been incorporated into tests described previously in Chapter 4. In the tests mentioned below, $T_{k+1} = \alpha T_k$ ($\alpha = 0.85$) and $T_{k+1} = T_k / \beta$ ($\beta = 0.9$).

- T2-1.1: The above cooling/reheating scheme is called a geometric scheme in which α and β are both static. For the heating rate β, if the β is dynamic during the SA searching process, then this cooling/reheating will become the enhanced geometric scheme. This cooling/reheating would become the enhanced geometric scheme. The geometric scheme is designed to avoid the problem of local minima cycling originally. In this test, the experiments with static β and dynamic β need to be performed to verify which scheme is better. The factor β is updated using $\beta = \beta - \varepsilon$, where $\varepsilon = 0.1$ and the update only happens when the reheat times reach the reheat limits. This test is designed to answer Q2-1.4.

Other variant settings in a mediator

Another two factors to be tested are the solution pool size and the cycle size given to the exam agents.

- T2-2.1: For question Q2-2.1, the solution pool is believed to be worth investigating in order to find out the best value setting. The COHH framework should be programmed with a solution pool size equal to the number of exam agents, and also with a solution pool size that is double the number of exam agents. Both should be tested using the ITC2007 benchmarks.
- T2-2.2: Question Q2-2.2 concerns the number of the iterations given to the exam agents in each cycle. To answer this question, the COHH framework should be implemented with the iteration number set to 1, 5 and 10, and tested with the examination datasets from the ITC2007.

Exam agents

Although the basic cooperative Hyper-heuristic frameworks are implemented to be the same, the communication scheme could still affect the local search within them. Communication timing refers to the decision about when to send a request-help message to the mediator when stuck in the local search.

- T2-3.1: The exam agents should be implemented to send the request-help message to the mediator with the non-improvement limit set to 25% and 50% of the given iteration the limits for the local search. Tests are required on the examination datasets ITC2007 with both setting values to identify the base setting. This test is designed to answer Q2-3.1.

Hypothesis-related experiments

For the COHH framework introduced in Chapter 5, five hypotheses are proposed based on the analysis of the algorithms and understanding of the cooperative working scheme. Targeting the COHH related hypotheses, six tests have been designed and are described below.

- T2-4.1: When selecting the best local solution from the solution pool, the selection between solutions with the same evaluation value should be implemented both with and without priority guidance for experimental purposes. Priorities are given to the exam agents. Priority-greedy selection should be compared with random-greedy selection for the solution pool through experiments on solving the ITC2007 benchmark dataset to verify H2-1.
- T2-4.2: The experimental results of the COHH framework should be categorized based on the number of agents. Both asynchronous and synchronous searches should be included for comparison. Comparing these results enables us to determine whether the number of agents is related to the performance of the COHH framework in solving exam timetabling problems. Specifically, the experimental results should be divided according to whether the COHH is based on four or six agents. This test is designed to verify H2-2.
- T2-4.3: The COHH framework experimental results should be categorized based on the cooperative search scheme for comparison in order to determine whether the asynchronous search generally outperforms the synchronous search. This test is designed to verify H2-3.

- T2-4.4: For the 6-agent-based COHH framework and the 4-agent-based COHH framework, both synchronous and asynchronous cooperative searches should be implemented and tested on the ITC2007 benchmark examination dataset. The coding of this COHH framework combination uses multi-thread to enable parallel searching of all the exam agents. This experiment is designed to verify H2-4.
- T2-4.5: The experimental results generated using the best COHH setting should be collected and compared to other, previously published best results from studies using different techniques to solve ITC20007 problems. This experiment is designed to verify H2-5.
- T2-4.6: The experimental results generated using the best COHH setting should be collected and compared to other, previously published best results from studies using other Hyper-heuristics to solve ITC20007 problems. This experiment is to verify H2-5.

6.5 Experimental Results

In this section, the experimental results of the tests described in Section 6.4 are presented and analysed. In the tables that follow, the bold number refers to the best results in the corresponding row of the tables. **BEST** in the tables refers to the best result within the given run in the test, while **AVE** refers to the average value of all the solution scores within the given run in the test.

6.5.1 RL-SA-HH

In the following tests in 6.5.1, unless other mentioned, all the tests were performed using datasets 7, 9, 11 and 12 in the ITC2007 benchmark. Each of the settings were tested for 10 runs on each of the four selected datasets to gather best result and average result for comparison purpose.

6.5.1.1 SA parameter tuning experiments and results

In the tests in this section for SA, a simple random selection method and sum utility update method are used during the implementation of RL-SA-HH. The following tests are conducted to improve the parameter tuning of the proposed RL-SA-HH and answer the related questions proposed previous.

Cooling rate test

The following tests were implemented in accordance with Test Design T1-1.1 to answer question Q1-1.1 by determining the best cooling rate. The proposed RL-SA-HH framework was implemented and tested with two different cooling rate settings of 0.5 and 0.85 respectively.

Table 31:Cooling Rate Test

Dataset	cooling rate = 0.5		cooling rate = 0.85	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	8952	8878	8915	**8867**
Exam 9	1493	1752	1419	**1352**
Exam 11	35600	35604	35611	**35504**
Exam 12	6054	**5960**	6074	5965

On the evidence presented in Table 31 above, the proposed framework performs better with the cooling rate set to 0.85 than to 0.5. Thus, the answer to the question Q1-1.1 is that the best cooling rate is 0.85. The implementation uses no reheating scheme and the reheating frequency is set to 10000.

Reheating rate test

The following test is implemented in accordance with test design T1-1.2 to determine the reheating rate and thereby to answer question Q1-1.2. The proposed RL-SA-HH framework was implemented and tested with two different reheating rates of 0.9 and 0.99 respectively.

Table 32:Reheating Rate Test

Dataset	reheating rate = 0.99		reheating rate = 0.9	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	8915	8685	8715	**8578**
Exam 9	1369	1457	1395	**1352**
Exam 11	35582	35604	35582	**35504**
Exam 12	6055	**5960**	6027	**5960**

On the evidence presented in Table 32 above, the reheating rate set to 0.9 outperforms the reheating rate at 0.99. The cooling rate used in this test is 0.85. The implementation using simple reheating scheme and the reheating frequency is set to 10000. No further action was taken when the reheating limit was reached. The answer to the question Q1-1.2, therefore, is that the best reheating rate is 0.9.

Reheating frequency test

The following test was implemented in accordance with'test design T1-1.3 to determine the best reheating frequency, and thereby to answer Q1-1.3.In the literature, the suggested Simulated Annealing value is usually 10000. However, in this approach, to help avoid getting caught within local optima, the reheating times are reduced to 1000, which has been verified as effective via experimentation. Both reheating frequency, 1000 and 10000 are tested to identify the best reheating frequency value setting.

Table 33:Reheating Frequency Test

Dataset	reheat frequency = 1000		reheat frequency = 10000	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	8631	**8497**	8858	8805
Exam 9	1347	**1278**	1352	1300
Exam 11	35782	**35266**	35704	35611
Exam 12	5985	5965	6034	**5911**

The results presented in Table 33 above suggest that the reheating frequency of 1000 is generally better than the reheating frequency of 10000. Therefore, in the following test, the reheat frequency is set as 1000. The cooling rate is set to 0.85 and the reheating rate is set to 0.9 in this test. The test result has answered the question Q1-1.3 by demonstrating that the best reheating frequency is 1000.

Reheating limit test

The following test was implemented in accordance with test design T1-1.4 to determine the best reheating limits value between 5 and 20 and thereby to answer Q1-1.4. The proposed RL-SA-HH framework is implemented and tested with different reheating limits of 5 and 20 respectively.

Table 34:Reheating Limit Test

Dataset	reheating limit = 5		reheating limit = 20	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	7890	**7528**	7898	7984
Exam 9	1520	**1506**	1519	**1506**
Exam 11	35673	**35375**	35764	35601
Exam 12	5985	**5911**	5987	5965

On the evidence presented in Table 34 above, the reheating limit of 5 outperforms the reheating limit of 20. Thus, the answer to the question Q1-1.4 is that the best reheating limit value is 5. The implementation uses the reheating scheme and the reheating frequency is set to 1000. The cooling rate is set to 0.85 and the reheating rate is set to 0.9 in this test. When the reheating limit is reached, the temperature is updated using T=(T0-T)/2 and the fbest is updated using fbest= fbest*1.1. Here T0 refers to the initial temperature value, T refers to the current temperature value and fbest refers to the evaluation value of the current best solution.

Resetting utility value test

The following test was implemented in accordance with test design T1-1.5 to identify whether resetting the low-level heuristics value improves performance, and thereby to answer Q1-1.5.

The utility values of the low-level heuristics can become similar after a certain number of iterations, which can adversely affect the selection methods and render the process ineffective. Two approaches were tested: resetting the utility value of the low-level heuristics when the maximum reheating point has been reached, and not resetting the utility

value. In this test, the framework is implemented uses a simple random selection method and sum utility update method.

Table 35:Resetting Utility Value Test

Dataset	reset		no reset	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	7530	**6721**	7890	7528
Exam 9	1340	**1320**	1520	1506
Exam 11	35733	**35519**	35673	35375
Exam 12	5995	**5955**	6015	6165

The results presented in Table 35 above indicate that in general the performance is improved by resetting the utility value of all low-level heuristics on reaching the maximum number of reheating iterations. Thus, the answer the question Q1-1.5 is that resetting the utility value of the low-level heuristics upon reaching the maximum reheating limit is better for the overall performance of the framework. Therefore, this approach was adopted in the selection method tests outlined in the next section. The implementation uses the reheating scheme where the reheating limit is set to 5 and the reheating frequency is set to 1000. The cooling rate is set to 0.85 and the reheating rate is set to 0.9 in this test.

6.5.1.2 Experiments and results relating to heuristic selection parameter tuning

In the test in this section, the implementation uses the SA setting with the reheating limit set to 5, the reheating frequency is set to 1000, the cooling rate is set to 0.85 and the reheating rate is set to 0.9. The reheating limiting scheme includes an increase in temperature, an increase in the current best solution evaluation value and a resetting of the utility values of the low-level heuristics.

ε-greedy ε value setting test

The following test is implemented in accordance with test design T1-2.1 to determine the best ε value for the ε-greedy selection method, and thereby to answer Q1-2.1.

The ε-greedy was tested with two different ε values in order to determine the best setting. The proposed framework was implemented with ε set to 0.01 and with ε set to 0.1 and tested to identify the best value setting for ε.

Table 36: ε -Greedy ε Value Setting Test

Dataset	ε = 0.01		ε = 0.1	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	8588	8381	8578	**7953**
Exam 9	1506	1343	1516	**1306**
Exam 11	36500	**35038**	36408	35281
Exam 12	5965	**5965**	5975	**5965**

The utility update method implemented in this test is the sum utility value method. On the evidence presented in Table 36 above, the ε-greedy selection method performs better when ε is set to 0.1. Therefore, the answer to question Q1-2.1 is that the best ε value setting is 0.1 in the ε-greedy selection method.

Softmax temperature setting test

The following test is implemented in accordance with test design T1-2.2 to determine the best value setting of temperature factor T for the Softmax selection method, and thereby to answer Q1-2.2.

The Softmax was tested with two different temperature T values in order to determine the best setting. The proposed framework was implemented with T set at 0.1 and with T set at 1 and then tested to identify the best value setting for T.

Table 37:Softmax T Value Setting Test

Dataset	T = 0.1		T = 1	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	8915	**8856**	8958	8912
Exam 9	1300	**1276**	1396	1396
Exam 11	35833	35832	35890	**35441**
Exam 12	6067	6065	6102	**5965**

The utility update method implemented in this test is the average utility value method. The temperature factor T controls the balance of exploration and exploitation. The test temperature values of 0.1 and 1 were chosen based on previously published test results, which suggest that setting this value to 0.1 on average improves performance. Although the advantage is modest within the best results shows in Table 37, after taking the average experimental result into consideration, it is believed that the Softmax selection method performs better when the temperature factor value is set to 0.1. Thus, the answer to the question Q1-2.2 is that the best value setting for temperature factor T is 0.1.

Selection method comparison test

The following test was implemented based on test design T1-2.3 to determine the Heuristic Selection Method to use for the proposed RL-SA-HH framework, and thereby to answer Q1-2.3.

As described above, five different selection methods were designed, some with differing operators. Tests were performed on all the selection methods with different

operator values. These tests use the sum utility update method. The best results for each selection method are presented in Table 38 below.

Table 38:Selection Method Comparison Test

Dataset	random	equal sequence	greedy	ε-greedy, $\varepsilon = 0.1$	ε-decay-greedy	Softmax, Temperature = 0.1
	BEST	**BEST**	**BEST**	**BEST**	**BEST**	**BEST**
Exam 7	8764	8644	7792	7953	**6501**	8856
Exam 9	1372	1372	1303	1306	1288	**1276**
Exam 11	35855	35840	35295	35281	**34054**	35832
Exam 12	6065	6065	**5965**	**5965**	**5965**	6065

In this test for the two ε-greedy values, the experimental results of ε set to 0.1 are presented. For the Softmax selection method, the temperature T setting of 0.1 is included. For comparative purposes, the result for the random selection method is included in Table 38. Clearly random selection and equal sequence, the two selection methods which do not use learning, perform the worst compared to the other heuristic selection methods.

Overall, the ε-decay-greedy method has the lowest best result over the chosen datasets. Thus, the answer to Q1-2.3 is that the best heuristic selection is the ε- decay-greedy method.

6.5.1.3 Experiments and results related to utility update method parameter tuning

The test described in this section was implemented using the SA setting with the reheating limit set to 5 and the reheating frequency set to 1000. In this test, the cooling rate is set to 0.85 and the reheating rate is set to 0.9. The reheating limiting scheme includes an increase in the temperature, an increase in the evaluation value of the current best solution and a

resetting of the low-level heuristics utility values. The heuristic selection method used in this test is the ε-decay-greedy heuristic selection method.

Discount utility method (discount factor value setting) test

The following test was implemented in accordance with test design T1-3.1 to find out the best discount factor r value setting for the discount sum utility method, and thereby to answer Q1-3.1. The discount utility method is affected by discount r values setting. The proposed framework was implemented with different discount r value set at 0.9 and r set to 0.5 and then tested to identify the best value setting for r.

Table 39:Discount Utility Method (Discount Factor Value Setting) Test

Dataset	**r = 0.9**		**r = 0.5**	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	7019	**6721**	7328	6967
Exam 9	1459	**1258**	1496	1359
Exam 11	36500	**34390**	35898	34390
Exam 12	5970	**5883**	5968	6013

As shown in Table 39 above, the discount utility method performs better when r is set to 0.9. Thus, the answer to the question Q1-3.1 is that the best value setting for the discount factor r is 0.9 for the discount sum utility method.

Temporal Difference utility method (step size value setting) test

The following test was implemented in accordance with test design T1-3.2 to determine the best step size of the Temporal Difference utility method, and thereby to answer Q1-3.2. The Temporal Difference utility method was tested with two different step size t values(1 and 5) in order to determine the best setting.

Table 40:Temporal Difference Utility Method (Step Size Value Setting) Test

Dataset	t=1		t=5	
	AVG	BEST	AVG	BEST
Exam 7	10023	9233	10220	**8733**
Exam 9	1427	**1285**	1973	1423
Exam 11	39566	34399	39769	**34397**
Exam 12	6567	**5911**	6560	5916

On the evidence presented in Table 40 above, there is little obvious difference in the performance of the Temporal Difference utility method when the step size is set to either 1 or 5. Therefore, in the following comparison test, the step size is set to 1 as it is believed to be able to speed up the searching process. This is because when the step size is set to 5, the utility value needs to calculate the estimated reward in the next 5 iterations, while when the step size is set to 1, only one further step is required (reducing the timing cost). From table 40, it can be seen that both step size setting get 2 beating result on the four tested datasets. However, when step size set to 1, three out four average result are lower when step size set to 5. In light of the above analysis, the answer to the question Q1-3.2 is that 1 is the best step size value.

Utility method comparison test

The following test was implemented based on test design T1-3.3 to identify the best Utility Update Method, and thereby to answer question Q1-3.3.

This test uses the ε-decay-greedy selection method combined with several different utility update methods. For this test, the Temporal Difference utility update method uses an estimation step size setting of 1 to establish the effectiveness of this approach. The experimental results of the discount utility method included in Table 41 below were generated with the discount factor set to 0.9.

Table 41:Utility Method Comparison Test

Datasets	Sum	Average	Discount	Temporal Difference
	BEST	**BEST**	**BEST**	**BEST**
Exam 7	6501	8824	**6721**	9233
Exam 9	1288	1370	**1258**	1423
Exam 11	34954	35529	**34390**	34399
Exam 12	6065	6065	**5883**	5916

It is clear from the results that the discount sum utility update method with the discount factor set to 0.9 performs best over the chosen datasets as shown in Table 41. The answer to the question Q1-3.3 therefore is that the best utility update method is the discount sum utility update method with the discount factor set to 0.9.

6.5.1.4 Hypotheses verification tests

This section describes the tests that were implemented to verify the related hypotheses regarding the proposed RL-SA-HH framework.

Reheating scheme test in SA

The following test is implemented based on test design T1-4.1 in order to verify hypothesis H1-1 that SA with the reheating scheme performs better in solving exam timetabling problems.

The proposed framework was implemented both with and without the reheating scheme. In the test described in this section, the utility value method is the sum utility update method while the heuristic selection method is the random greedy selection method. For the move acceptance Simulated Annealing algorithms, the cooling rate is set to 0.85, the reheating rate is set to 0.9 and the reheating frequency is set to 10000 with a

reheating limit value of 5. No further action was taken when the reheating limit was reached. Both schemes, with and without reheating, were tested to identify the best reheating scheme.

Table 42:Reheating Scheme Test in SA

Dataset	with reheating		without reheating	
	AVG	BEST	AVG	BEST
Exam 7	8858	**8805**	8915	8867
Exam 9	1352	**1300**	1419	1352
Exam 11	35704	35611	35611	**35504**
Exam 12	6034	**5911**	6074	5965

The results presented in Table 42 above prove that the hypothesis H1-1 is correct: the reheating scheme enables the RL-SA-HH to perform better research.

Reheating limit control test in SA

The following test was implemented in accordance with test design T1-4.2 in order to verify hypothesis H1-2 that reheating limits help SA to perform better.

The proposed framework was implemented both with and without the reheating scheme. In the test in this section, the utility value method is the sum utility update method while the heuristic selection method is the random greedy selection method. For the move acceptance Simulated Annealing algorithms, the cooling rate is set to 0.85, the reheating rate is set to 0.9 and the reheating frequency is set to 10000, with a reheating limit value of 5. When the reheating limit is reached, the temperature increases, the current best solution evaluation value increases and the utility values of low-level heuristics are reset.

The reheating scheme was tested both with and without a reheating limitation to identify the best reheating limit control setting. The results presented in Table 43 prove

that the hypothesis H1-2 is correct: controlling the reheating limit enables the RL-SA-HH to perform better research.

Table 43:Reheating Limit Control Test in SA

Dataset	With reheating limit		Without reheating limit	
	AVG	**BEST**	**AVG**	**BEST**
Exam 7	8631	**8597**	8858	8805
Exam 9	1347	**1278**	1352	1300
Exam 11	35782	**35466**	35704	35611
Exam 12	5985	5965	6034	**5911**

RL-SA-HH capability test

The following test was implemented in accordance with test design T1-4.3 in order to verify hypothesis H1-3 that RL-SA-HH is capable of solving complex exam timetabling problem benchmarks.

Having chosen the best selection method (ε-decay-greedy) and the best utility update method (discount sum utility), the test was conducted over all 12 datasets from the ITC2007 benchmarks, and the best evaluation value was recorded for 10 times for each dataset. A comparison of the current best techniques in the literature is presented in Table 44 below.

Table 44 shows the result of using RL-SA-HH to solve the ITC2007 exam timetabling problems. The proposed RL-SA-HH managed to solve all the problems. It therefore can be argued that the proposed approach is applicable and worthy of further research and experimentation. This approach managed to solve all the ITC2007 datasets despite their complexity. Therefore, hypothesis H1-3 can be verified: the RL-SA-HH framework is a generally applicable approach for solving exam timetabling problems.

Comparison of RL-SA-HH and other techniques

The following test was implemented in accordance with test design T1-4.4 in order to verify hypothesis H1-4 that the proposed RL-SA-HH framework is competitive. The best results generated by using the proposed RL-SA-HH framework were compared with the results from five different approaches described in the literature. The results of this comparison are presented in Table 44 below.

Table 44:Comparison of RL-SA-HH and Other Techniques

Dataset	RL-SA-HH		Other Techniques				
	AVG	BEST	DSO (Hamilton-Bryce et al., 2014)	Hyper-heuristic (M¨uller, 2009)	Adaptive liner combination (Rahman et al., 2014)	Graph colouring (Sabar et al., 2012)	Multistage algorithm (Gogos et al., 2011)
Exam 1	6059	6059	5186	**4370**	5231	6234	5814
Exam 2	863	863	405	400	433	**395**	1062
Exam 3	14027	14027	9399	10049	**9265**	13302	14179
Exam 4	20031	20031	19031	18141	**17787**	17940	20207
Exam 5	3637	3637	3117	**2988**	3083	3900	3986
Exam 6	26910	26910	**26055**	26585	26060	27000	27755
Exam 7	6572	6572	**3997**	4213	10712	6214	6885
Exam 8	10485	10485	**7303**	7742	12713	8552	10449
Exam 9	1267	1267	1048	**1030**	1111	N/A	N/A
Exam 10	**14357**	**14357**	14789	16682	14825	N/A	N/A
Exam 11	34054	34054	30311	34129	**28891**	N/A	N/A
Exam 12	5509	5509	**5369**	5535	6181	N/A	N/A

The results obtained show that the proposed method is reasonably competitive, and in fact has achieved the best result for Exam 10 and is approaching the best for several others. It therefore can be argued that the approach taken is promising and worthy of further research and experimentation. Although the proposed approach did not generate best results for any of the other datasets, it should be noted that it did manage to solve all the ITC2007 datasets despite their complexity.

Additionally, the proposed approach is capable of delivering relatively good solutions compared to other techniques in general. Examination of the penalty values for the generated results shows that the proposed approach performs better for those datasets with a smaller size, such as Exams 10, 6 and 9, than for the datasets with a larger size, such as Exams 7 and 3. It is reasonable to suggest that this is due to the smaller datasets having been allowed a longer learning time than the larger datasets. Therefore, hypothesis H1-4 can been verified: the proposed framework is relatively competitive but not optimal.

Comparison of RL-SA-HH and other HH techniques

The following test was implemented in accordance with test design T1-4.5 in order to verify hypothesis H1-5, that RL-SA-HH is competitive in relation to solving exam timetabling problems compared to other Hyper-heuristics, described in the literature, which have been used for the same purpose.

Table 45:Comparison of RL-SA-HH and Other HHs

Dataset	**RL-SA-HH**	**Other Techniques**			
	BEST	**HSHH (Anwar et al., 2013)**	**GCCHH (Sabar et al., 2012)**	**EAHH (Pillay et al., 2010)**	**AIH (Burke et al., 2014)**
Exam 1	**6059**	11823	6234	8559	6235
Exam 2	863	976	**395**	830	2974
Exam 3	14027	26670	13302	**11576**	15832
Exam 4	20031	\	**17940**	21901	35106
Exam 5	**3637**	6772	3900	3969	4873
Exam 6	**26910**	30980	27000	28340	31756
Exam 7	6572	11762	**6214**	8167	11562
Exam 8	10485	16286	**8552**	12658	20994
Exam 9	**1267**	-	-	-	-
Exam 10	**14357**	-	-	-	-
Exam 11	**34054**	-	-	-	-
Exam 12	**5509**	-	-	-	-

It is interesting to examine the experimental results of other Hyper-heuristic techniques which have been applied to solving examination timetabling problems using the ITC2007 benchmarks. A comparison with the approach undertaken in this work is useful to determine whether it improves on the current work using Hyper-heuristics.

The results shown in Table 45 above show that this approach has achieved best results for three of the datasets as compared to the other Hyper-heuristics. Indeed, for the other datasets, the results are relatively good (above average), suggesting that this work makes a positive contribution to improving research within the area of Hyper-heuristics. Therefore, it can be argued that hypothesis H1-5 is correct, since the RL-SA-HH delivered the best results of the various approaches applied to the ITC2007 datasets three times and managed to solve all 12 datasets.

6.5.2 COHH

In the following tests in 6.5.2, unless other mentioned, all the tests were performed using datasets 7, 9, 11 and 12 in the ITC2007 benchmark. Each of the settings were tested for 10 runs on each of the four selected datasets to gather best result and average result for comparison purpose.

6.5.2.1 Mediator agent tests

SA reheating scheme (geometric vs enhanced geometric) test

The following test is implemented in accordance with test design T2-1.1 in order to determine whether the geometric cooling/reheating scheme or the enhanced geometric cooling/reheating scheme is better, and thereby to answer question Q2-1.1.

This test compared the mediator agent with SA that uses a geometric reheating scheme and with SA that does not use the reheating scheme. The cooling rate in this test is 0.85 and the reheating rate is 0.9. For the enhanced geometric reheating scheme-based mediator agent implementation the update rate of the reheating rate is 0.1 and the update frequency is 5. The COHH frameworks (with both a 4-agent and a 6-agent design and using both a synchronous search and an asynchronous search) were tested, using both geometric and enhanced geometric schemes, to find the best reheating scheme setting.

Table 46:SA Reheating Scheme (Geometric vs Enhanced Geometric) Test

Dataset	Geometric				Enhanced Geometric			
	Syn search BEST		Asyn search BEST		Syn search BEST		Asyn search BEST	
	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent
Exam 7	5935	5416	5259	**4602**	5843	5407	5125	**4602**
Exam 9	1457	1483	1120	1179	1387	1338	1220	**1174**
Exam 11	45220	42461	38229	35817	45240	41641	38228	**35187**
Exam 12	6253	6813	5495	5496	6242	6188	5484	**5440**

From the results shown in Table 46 above, it is obvious that the mediator agent led the overall search more successfully with the enhanced geometric reheating scheme. The best results were generated from the asynchronous search with the 6-agent design in the enhanced geometric scheme. Also, the overall performance of the enhanced geometric reheating scheme is better than that of the geometric scheme. Thus, the answer to question Q2-1.1 is that the enhanced geometric reheating scheme is a better choice.

Solution pool size test

The following test was implemented in accordance with test design T2-2.1 in order to determine whether the best solution pool size is double or equal to the number of exam agents, and thereby to answer question Q2-2.1.

This test was designed to determine the best value setting for the solution pool size held by the mediator agent. The solution pool holds all the received local elite solution. The test compared the performance of the proposed framework when the size of the solution pool was equal to the number of agents and when it was double the number of agents.

Table 47:Solution Pool Size Test

Dataset	solution pool = number of agents				solution pool = (number of agents)*2			
	Syn search BEST		Asyn search BEST		Syn search BEST		Asyn search BEST	
	4-agent pool size=6	6-agent pool size=4	4-agent pool size=6	6-agent pool size=4	4-agent pool size=12	6-agent pool size=8	4-agent pool size=12	6-agent pool size=8
Exam 7	5929	5417	5152	**4502**	5843	5407	5125	4602
Exam 9	1456	1432	1219	1219	1387	1338	1220	**1174**
Exam 11	45215	42371	35229	35917	45240	41641	38228	**35187**
Exam 12	6233	6713	5496	5496	6242	6188	5484	**5440**

The COHH frameworks (with both a 4-agent and a 6-agent design and using both a synchronous search and an asynchronous search) were tested with the solution pool size set both as the number of agents and as twice the number of agents to identify the best solution pool size setting.

From the results shown in Table 47 above, it is obvious that implementing the COHH with a larger solution pool (twice the number of agents, in this test) improves its performance. It is believed that this is because the larger size gives the mediator agent more diversification enabling it to escape the local optimal cycle. The results provide an answer to question Q2-2.1: the best solution pool size is double the number of exam agents.

Cycle size test

The next test was implemented in accordance with test design T2-2.2 to determine the best cycle size between 1 and 10 for the COHH frameworks, and thereby to answer Q2-2.2. This test is designed to determine the best value setting for the size of the solution pool held by the mediator agent. The solution pool holds all the received local elite solutions.

This test compared the performance of the framework when the iteration limit was set to 1 and 10 respectively.

Table 48:Cycle Size Test

Dataset	cycle size = 1				cycle size = 10			
	Syn search BEST		Asyn search BEST		Syn search BEST		Asyn search BEST	
	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent
Exam 7	5869	5417	5151	**4502**	5843	5407	5125	4602
Exam 9	1441	1430	1217	1211	1387	1338	1220	**1174**
Exam 11	44079	42328	35193	35193	45240	41641	38228	**35187**
Exam 12	6170	6706	5490	5495	6242	6188	5484	**5440**

The COHH frameworks (with both a 4-agent and a 6-agent design and using both a synchronous and an asynchronous search) were tested with cycle size set to 1 and 10 respectively.

From the results shown in Table 48, it is clear that COHH performs better with the cycle size set at 10 than when set at 1. Giving more iterations to the exam agents gives them more freedom on local searches instead spend more time on the agents' communication. The answer to question Q2-2.2, therefore, is that, given a choice between 1 or 10, the best cycle size for the COHH frameworks is 10.

Exam agent parameter test - Request-help frequency test

The following test was implemented in accordance with test design T2-3.1 to identify the best request-help frequency between 5 and 2, and thereby to answer question Q2-3.1.

In this test, the iteration limit number given to exam agents is set at 10. Therefore, the request-help frequency when set to half the number of iterations is 5; when set to a quarter of the number of iterations, it is 2.

Table 49:Request-Help Frequency Test

Dataset	Request-help frequency = (iteration amount)/2 *frequency=5				Request-help frequency = (iteration amount)/4 *frequency=2			
	Syn search BEST		Asyn search BEST		Syn search BEST		Asyn search BEST	
	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent
Exam 7	5843	5407	5125	**4602**	6092	5495	5344	**4602**
Exam 9	1387	1338	1220	**1174**	1446	1297	1272	1417
Exam 11	45240	41641	38228	**35187**	47114	43241	38266	36991
Exam 12	6242	6188	5484	**5440**	6240	6506	5484	5965

The COHH frameworks (with both a 4-agent and a 6-agent design and using both a synchronous and an asynchronous search) were tested with the request-help frequency value set to 5 and 2 respectively.

From the results presented in Table 49 above, it is obvious that COHH performs better when the request-help frequency test is set to 5 than when it is set to 2. Thus the answer to question Q2-3.1 is that the best request-help frequency is 5.

6.5.2.2 Hypotheses verification tests

The test described in this section were implemented to verify the related hypotheses regarding the proposed COHH framework. In the experiments in the following sections, the SA in the mediator uses an enhanced geometric scheme with a solution pool size set to

twice the number of agents, the cycle size set to 10 and the request-help frequency within the exam agents set to 5 iterations.

Comparison of priority setting and random selection

The following test was implemented in accordance with test design T2-4.1 in order to verify hypothesis H2-1, which is that the priority setting for the exam agents performs better than random selection in the mediator agent when more than one solution with the same evaluation value is received.

As previously noted, the selection of two elite solutions with the same evaluation value from the solution pool can be priority guided or not. This experiment implemented both selection schemes and tested to identify the best priority setting.

The COHH frameworks (with both a 4-agent and a 6-agent design and using both a synchronous and an asynchronous search) were tested uqsing priority-greedy selection and random-greedy selection respectively. The comparison result in table 50 shows no clear advantage in using the priority greedy selection method. However, its results are no worse than the random selection method.

Table 50:Comparison of Priority Setting and Random Selection

Dataset	Priority-Greedy Selection				Random-Greedy Selection			
	Syn search BEST		Asyn search BEST		Syn search BEST		Asyn search BEST	
	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent	4-agent	6-agent
Exam 7	5843	5407	5125	4602	5843	5269	5119	**4597**
Exam 9	1387	1338	1220	1174	1464	1296	1220	**1172**
Exam 11	45240	41641	38228	**35187**	45240	43100	38228	**35187**
Exam 12	6242	6188	5484	**5440**	6225	6044	5484	5945

This result suggests that further research on the priority-greedy selection scheme is necessary. The priority given to the agents could be calculated based on the agents'

performance. A dynamic priority scheme could be an option for improving the performance of the priority selection method. Therefore, hypothesis H2-1 has been proved wrong.

In the following tests, the selection method between agents is a random-greedy selection method.

Comparison of 4-agent and 6-agent design

The following test was implemented in accordance with test design T2-4.2 in order to verify hypothesis H2-2, that the proposed COHH framework performs better with more exam agents.

Comparative results grouped by number of agents are shown in Table 52. For both the 6-agent and the 4-agent design, the performance of the synchronous scheme and the asynchronous scheme design in the mediator agent were identical. The test results were collected from 50 runs of the program in all combinations of the two agent designs and the two cooperative schemes mentioned above.

Table 51:Comparative performance of 4-agent and 6-agent designs

Dataset	6-agent based				4-agent based			
	Syn search		Asyn search		Syn search		Asyn search	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Exam 7	5563	5407	4735	**4602**	6012	5843	5299	5125
Exam 9	1390	1338	1219	**1174**	1441	1387	1267	1220
Exam 11	43140	41641	36453	**35187**	46868	45240	39604	38228
Exam 12	6297	6188	5536	**5440**	6352	6242	5581	5484

On the evidence presented in Table 51, the COHH framework with the 6-agent design (soft constraint-based) performs better than the COHH framework with the 4-agent design (multiple low-level heuristics sets). It is believed that the final performance is affected by the number of agents. Although the difference between the designs is subtle,

the results nevertheless prove hypothesis H2-2. The 4-agent designs performance could be further improved by adding exam agents.

Comparison of synchronous and asynchronous searches

The following test was implemented in accordance with test design T2-4.3 in order to verify hypothesis H2-3 that the COHH framework performs better with the asynchronous scheme than with the synchronous scheme.

Comparative results grouped by cooperative searching scheme are shown in Table 53 below. For both the synchronous and the asynchronous schemes, the 6-agent design and the 4-agent design were identical. The test results were collected from 50 runs of the program in all combinations of the two agent designs and the two cooperative schemes mentioned above.

Table 52:Comparison of Synchronous and Asynchronous Searches

Dataset	Synchronous Search				Asynchronous Search			
	6-agent based		4-agent based		6-agent based		4-agent based	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Exam 7	5563	5407	6012	5843	4735	**4602**	5299	5125
Exam 9	1390	1338	1441	1387	1219	**1174**	1267	1220
Exam 11	43140	41641	46868	45240	36453	**35187**	39604	38228
Exam 12	6297	6188	6352	6242	5536	**5440**	5581	5484

As Table 52 above clearly shows, the synchronous scheme performs worse than the asynchronous scheme in both the 4-agent design and the 6-agent design. A possible explanation is that the waiting time in each cycle for all exam agents in the synchronous scheme slows down the searching process, affecting the performance of the synchronous scheme. Hypothesis H2-3 is therefore correct: the asynchronous search performs better in both the 4-agent design and the 6-agent design.

Applicability test

The following test was implemented in accordance with test design T2-4.4 in order to verify hypothesis H2-4 that the proposed COHH frameworks are applicable to solving exam timetabling problems.

Two different cooperative search schemes combined with two different agent designs were implemented and tested on all 12 datasets from the ITC2007 benchmarks. The experimental results prove that the proposed COHH framework is capable of solving exam timetabling problems. The tests were run 50 times for each combination to gather the best results.

Table 53:Applicability Test

Dataset	Synchronous Search				Asynchronous Search			
	6-agent based		4-agent based		6-agent based		4-agent based	
	AVG	BEST	AVG	BEST	AVG	BEST	AVG	BEST
Exam 1	5609	5489	5562	**5440**	4842	**4748**	5239	4763
Exam 2	494	**479**	506	491	417	**408**	446	413
Exam 3	14183	13293	18017	16887	11513	**11233**	15196	14202
Exam 4	26400	24490	27432	25448	22172	**21527**	22956	21657
Exam 5	3422	3297	3990	3845	2870	**2773**	3492	3326
Exam 6	34698	31527	33626	**30553**	27840	**26515**	27866	26795
Exam 7	5563	5407	6012	5843	4735	**4602**	5299	5125
Exam 8	8494	8215	9913	9588	7390	**7106**	8345	8102
Exam 9	1390	1338	1441	1387	1219	**1174**	1267	1220
Exam 10	14208	13862	18324	17878	12704	**12157**	15519	15215
Exam 11	43140	41641	46868	45240	36453	**35187**	39604	38228
Exam 12	6297	6188	6352	6242	5536	**5440**	5581	5484

On the evidence presented in Table 53, it is obvious that the proposed COHH framework is capable of solving the ITC2007 exam timetabling problems with both synchronous and asynchronous search schemes and both 4-agent and 6-agent designs. Therefore, hypothesis H2-4 is justified. Through analysing the data in table 54, it is

oblivious that the cooperative scheme using 6 agent design outperforms the 4 agent design. It is believed that the result proved that the agent amount affects the overall performance of the cooperative search framework.

Comparison of COHH and other techniques

The following test was implemented in accordance with test design T2-4.5 in order to verify hypothesis H2-5, that the best results generated from the proposed COHH framework are competitive for solving the ITC2007 benchmark.

Table 54:Comparison of COHH and Other Techniques

Dataset	COHH	Other Techniques				
	BEST	Hyper-heuristic (M¨uller et al., 2009)	Adaptive Liner Combination (Rahman et al., 2014)	Graph colouring (Sabar et al., 2012)	Multistage algorithm (Gogos et al., 2011)	Hyper-heuristic (M¨uller et al., 2009)
Exam 1	4748	5186	**4370**	5231	6234	5814
Exam 2	408	405	400	433	**395**	1062
Exam 3	11233	9399	10049	**9265**	13302	14179
Exam 4	21527	19031	18141	**17787**	17940	20207
Exam 5	**2773**	3117	2988	3083	3900	3986
Exam 6	26515	**26055**	26585	26060	27000	27755
Exam 7	4602	**3997**	4213	10712	6214	6885
Exam 8	**7106**	7303	7742	12713	8552	10449
Exam 9	1174	1048	**1030**	1111	N/A	N/A
Exam 10	**12157**	14789	16682	14825	N/A	N/A
Exam 11	35187	30311	34129	**28891**	N/A	N/A
Exam 12	5440	**5369**	5535	6181	N/A	N/A

The best results from using the implementation of the proposed COHH framework were collected and compared with the best results from using other techniques described in the literature. The results of this comparison, which are presented in Table 54 below shows that the COHH framework produced the best result for three of the 12 databases, and relatively good results for all the others. This supports the hypothesis that the COHH

framework is competitive when implemented in solving exam timetabling problems. Therefore, it can be argued that hypothesis H2-5 is correct.

Comparison of proposed frameworks with other sequential hyper-heuristics for ITC2007

The following test was implemented in accordance with test design T2-4.6 in order to verify hypothesis H2-6 that cooperative schemes can be used to improve the performance of the Hyper-heuristic.

The best results collected when using the implementation of the proposed COHH framework (using the 6-agent design) to solve the ITC2007 exam timetabling problems were also compared to the best results achieved by using various other sequential Hyper-heuristics described in the literature.

Table 55:Comparative Performance of COHH, RL-SA-HH and Other HH/Cooperative Techniques

Dataset	COHH	RL-SA-HH	Other Techniques			
	BEST	BEST	HSHH (Anwar et al., 2013)	GCCHH (Sabar et al., 2012)	EAHH (Pillay et al., 2010)	AIH (Burke et al., 2014)
Exam 1	**4748**	6059	11823	6234	8559	6235
Exam 2	408	863	976	**395**	830	2974
Exam 3	**11233**	14027	26670	13302	11576	15832
Exam 4	21527	20031	////	**17940**	21901	35106
Exam 5	**2773**	3637	6772	3900	3969	4873
Exam 6	**26515**	26910	30980	27000	28340	31756
Exam 7	**4602**	6572	11762	6214	8167	11562
Exam 8	**7106**	10485	16286	8552	12658	20994
Exam 9	**1174**	1267	-	-	-	-
Exam 10	**12157**	14357	-	-	-	-
Exam 11	**35187**	34054	-	-	-	-
Exam 12	**5440**	5509	-	-	-	-

The results presented in Table 55 show that the COHH framework produced the best result for six of the 12 databases, and relatively good results for all the others. Moreover, the

proposed approach managed to solve all of the 12 datasets. When the proposed COHH framework is compared with the proposed RL-SA-HH (both from this thesis), the advantage is obvious. Therefore, it can be argued that a cooperative searching scheme is an effective approach that can be used to improve the performance of the usual 2-stage Hyper-heuristic. Hence, hypothesis H2-6 is proved.

6.6 Summary

In this chapter, the implementation of the RL-SA-HH framework and the COHH framework is described. As part of the implementation process, related questions and hypotheses were proposed. Adequate experiments were performed to verify these questions and hypotheses. These experiments mainly involved testing variant settings for the framework components and comparative testing to prove the capability of the proposed framework.

As part of the RL-SA-HH implementation process, all the proposed heuristics selection methods, as well as the utility value method inspired by the Reinforcement learning algorithms, were tested with the ITC2007 benchmark datasets. After adequate testing, the best selection method is believed to be the ε-decay-greedy selection method, while the best utility update method is the discount utility update method. Specifically, the discount utility update method performs better with the discount factor set to 0.9.

As a vital component to the Hyper-heuristic, the move acceptance Simulated Annealing algorithm was also tested to identify the best cooling and reheating scheme. The experimental results suggest that the implemented Hyper-heuristic performs best when the cooling rate is set to 0.85, the reheating rate is set to 0.9 and the reheating frequency is set

to 1000 with a reheating limit of 5. The implemented RL-SA-HH algorithms with the best settings were used to solve the examination problems from the ITC2007 benchmark. The final best results were compared to the experimental results generated by other techniques from the literature that have been used to solve ITC2007 exam scheduling problems. The comparison proves that the proposed RL-SA-HH framework is applicable and competitive, especially when compared to the other sequential Hyper-heuristics in the literature.

In the COHH-related test, the COHH framework was implemented with both a 4-agent and a 6-agent design, using both synchronous and asynchronous searches. As part of the implementation process, the Simulated Annealing algorithm in the mediator agent was also tested. The idea of applying' the enhanced geometric reheating scheme is supported by the experimental results. Also, the solution pool size used in the mediator agent was proven to be better when set as twice the number of agents, while the request-help frequency of the exam agents proved to be better when set as half the given iteration limit. Although the hypothesis related to the messages handling scheme proves to be wrong that the random-greedy-selection approach actually outperform the priority based selection. Further research should be focused on the improve of the priority allocation approach to each exam agents.

Furthermore, using the best settings and both a 4-agent and a 6-agent design, the synchronous search and the asynchronous search were compared, and the asynchronous search was shown to perform better. Only a minor difference was evident between the performance of the 6-agent and 4-agent designs.

In the next chapter, the work that has been done thus far and reported in this thesis will be summarized and concluded. The advantages and disadvantages of the thesis will also be presented, together with suggestions for the direction of potential future work.

CHAPTER 7. CONCLUSIONS

7.1 Objectives

Four main objectives were established for and have been presented in this thesis. The first and main objective was to develop a general Hyper-heuristic approach that can be used to solve complicated exam timetabling problems. The second objective was to demonstrate the learning ability of the proposed Hyper-heuristic with Reinforcement Learning. Having done so, the third objective was to further improve the proposed Hyper-heuristic framework with the cooperative scheme. Last but not least, the thesis set out to demonstrate that the proposed framework has the flexibility to adapt to exam timetabling datasets with differed characteristics. The future research of the hyper-heuristic is believed to be the generality which is the focus of this research.

7.2 Achievements and Contributions

7.2.1 Achievements

To achieve the objectives described above, the research was conducted in three stages as follows.

The first stage involved the development of an online Hyper-heuristic framework. A suitable Hyper-heuristic framework was designed to target exam timetabling problems.

The move acceptance of the Hyper-heuristic framework was implemented with Simulated Annealing algorithms. In addition, 19 low-level heuristics were designed and implemented for the Hyper-heuristic framework to use.

The second stage focused on improving the learning ability of the implemented Hyper-heuristic framework. Learning ability determines the generality of the proposed Hyper-heuristic framework. Enabling the framework to learn means that it can be reused to solve different timetabling problems or even be applied to different scheduling and optimization problems. Research on the learning ability of the proposed Hyper-heuristic was conducted by designing multiple heuristic selection methods and reward schemes using Reinforcement Learning algorithms. The intelligent heuristic selection process, with the help of suitable reward schemes, enables the proposed Hyper-heuristic to track the performance of the low-level heuristics and learn from them. This tracking process enables the proposed Hyper-heuristic to solve problems by operating upon various low-level heuristics without knowledge of the targeted problems.

The third stage of the research involved further improvement of the performance of the proposed Hyper-heuristic framework by using a cooperative scheme. A cooperative scheme is used to manage the cooperation between multiple Hyper-heuristic search agents. In a well-designed Hyper-heuristic framework, performance could also be affected by the consistent use of low-level heuristics. The cooperative scheme managed to use multiple Hyper-heuristics with the same framework but with different low-level heuristics. The learning/training within the Hyper-heuristic was minimized and the efficiency of the overall cooperative Hyper-heuristic was improved as a result.

Reinforcement Learning algorithms were successfully combined with the Hyper-heuristic. In the development of the Reinforcement Learning-Simulated Annealing-Hyper-heuristic framework, five different heuristic selection methods inspired by Reinforcement Learning algorithms were designed and implemented. They are equal sequence, greedy, ε-greedy, ε-decay-greedy and Softmax selection. Of these, the ε-decay-greedy method performs the best. In addition, four types of reward scheme inspired by Reinforcement Learning algorithms were designed and implemented: sum, average, Discount and Temporal Difference. Of these, the Discount reward scheme performs the best. All these methods were implemented with the aim of establishing the best learning scheme for the proposed framework. These above methods have all been tested and proved to work, thereby justifying the idea of bringing Reinforcement Learning into the Hyper-heuristic framework.

A cooperative search scheme has been successfully brought into the proposed Hyper-heuristic framework. In the development of the cooperative Hyper-heuristic framework, two different cooperation/communication schemes were proposed and implemented: synchronous and asynchronous. In addition, two different agent designs for the cooperative framework were proposed and implemented: six agents with different evaluation functions and four agents with different low-level heuristics. The two cooperation schemes were both tested with the same agent design in order to determine the best communication scheme, and the asynchronous scheme was shown to outperform the synchronous scheme. Using the same asynchronous scheme, the two agent designs were then tested and compared, and the 6-agent design was shown to perform better. The above

methods have all been tested and proved to work, thereby justifying the idea of bringing the cooperative scheme into the Hyper-heuristic framework.

The learning ability of the Hyper-heuristic approach has been improved successfully. Once the best heuristic selection method and the best reward scheme were selected, the best component setting of the proposed RL-SA-HH was decided. This RL-SA-HH was compared to the best results generated using other Hyper-heuristics approaches described in the literature and was shown to outperform them. The proposed RL-SA-HH managed to solve all 12 datasets from the ITC2007 benchmark. Also, as part of the comparison test of the heuristic selection methods, all the proposed heuristic selection methods were compared with random selection methods. The random selection method was found to perform worse than the heuristic selection with learning ability. Therefore, it is believed that the learning ability of the Hyper-heuristic approach has been improved as well as its performance.

The cooperative scheme has successfully improved the performance of the Hyper-heuristic framework. Although the purpose of the research was to provide an approach with high generality rather than one which outperforms all other approaches described in the literature, the proposed RL-SA-HH approach managed to deliver seven best results compared to other, previously published Hyper-heuristic approaches. However, its performance is not sufficiently competitive when compared to that of other non-Hyper-heuristic approaches described in the literature, only one of which it managed to outperform.

As for the proposed COHH approach, it managed to deliver three best results when compared to all previous approaches and 10 best results when compared to previous Hyper-

heuristic approaches. It can therefore be concluded that the performance of the proposed Hyper-heuristic approaches has been improved using the cooperative search scheme.

7.3 Contribution and Discussion

The proposed RL-SA-HH and COHH frameworks can be described as truly generalized. In the literature, work such as directed selection optimisation method proposed by Hamliton-Bryce et al. (2014), adaptive linear combination method proposed by Rahman et al. (2014), multistage algorithm proposed by Gogos et al. (2011) has all managed to generate several competing results on the ITC2007 benchmark. However, these works are all rather problem dependent compare to hyper-heuristic approaches which separate the problem domain and the heuristic selection domain. For the proposed RL-SA-HH approach, the framework is ready to use for any new timetabling problems with suitable low-level heuristics and evaluation functions. There is no need for any parameter tuning by users of the proposed framework (higher level). For example, in a real-world academic institution, the framework can be easily applied to different faculties even if the constraints vary. The only work required is to update the evaluation function 'according to the various constraints. As for other scheduling problems, with the construction methods, evaluation functions and low-level heuristics available, this proposed framework can be reused with little human intervention.

The generality of the proposed COHH approach can also be argued as the communication stage and the higher level of the Hyper-heuristic agents are not performing searches on the target problems directly. In the literature, Ouelhadj and Petrovic (2008) proposed a cooperative hyper-heuristic framework. In their work, the low-level heuristics

are designed as agents which can perform low-level searches in parallel. In the cooperative hyper-heuristic proposed in this research, the learning is in parallel instead. This research managed to allow multiple different hyper-heuristics to perform the search in parallel and learning within each hyper-heuristic is autonomous and independent. This research improves both the generality and the performance of the hyper-heuristic approaches at a new level.

In both proposed frameworks, the learning targets the performance of the low-level heuristics. The value of learning is the result of the performance evaluation of each low-level heuristic in each iteration. Through the test, the ε-decay-greedy selection method performs the best in the learning process using the feedback from the low-level heuristics. In this thesis, the learning ability has been proved to be vital to improving the performance of the Hyper-heuristic (by comparing learning selection methods to random selection methods). The success of using Reinforcement Learning in the heuristics selection process proves that Machine Learning approaches can be used to solve timetabling problems easily. As is obvious from the results of the test, the Hyper-heuristic framework performs worse when compared with other non-Hyper-heuristic approaches, which are believed to be more problem-specific. In order to preserve generality and further improve performance, the cooperative scheme was brought into the research of Hyper-heuristic. The cooperative search scheme was proved to be a great approach for achieving balance between performance and geniality. The cooperative framework does not affect the Hyper-heuristic framework within the agents.

The two proposed frameworks can be easily applied to other timetabling problems without parameter tuning. This feature improves the independence of the proposed

approaches from the targeted problems. Furthermore, the successful use of Reinforcement Learning creates a new, simple way to use complex Machine Learning to solve complicated timetabling problems. The Hyper-heuristic framework features high flexibility and is easily embedded with other techniques without knowledge of the target problems. It is also argued that the pattern of Reinforcement Learning plus Hyper-heuristic framework contributes to the generality and simplifies the research performed by Machine Learning approaches. The proposed cooperative Hyper-heuristic also has the flexibility to combine non-Hyper-heuristic approaches with Hyper-heuristic approaches.

The move acceptance must be well designed to escape the local optimal. During the development of the RL-SA-HH approach, the move acceptance used Simulated Annealing. The Simulated Annealing move acceptance was implemented without a reheating scheme initially and did not perform well. It is believed that, as the search proceeds, the random acceptance of a worse solution is reduced, which could result in getting caught by local optimal. Therefore, the move acceptance was revised to include a complicated reheating scheme, and a limitation on the number of reheating times to trigger bigger reheating actions. As a result, the performance of the RL-SA-HH improved.

For the cooperative Hyper-heuristic framework, it is obvious that the asynchronous search outperforms the synchronous search. Although an attempt was made to improve the synchronous search by accepting multiple moves in one circle, the results were not as expected. Future work should focus on improving the asynchronous search rather than the synchronous search.

7.4 Future Work

Future research in relation to the development of the Reinforcement Learning Simulated Annealing Hyper-heuristic framework should focus on performance analysis and improvement of the low-level heuristics module. The initial construction approaches could also be improved to enhance the performance of the Hyper-heuristic. In this thesis, the main focus has been on the upper-layer design of the Hyper-heuristic framework rather than the low-level heuristic components. In this thesis, all 19 low-level heuristics are offered for the heuristic selection layer to select. The possibility of exploring a larger search space could be increased by adding more different low-level heuristics. Also, the connection between the number and the type of the available low-level heuristics and the overall performance of the Hyper-heuristic framework requires further research.

The Temporal Difference reward function could also have potential if the step size used for predicting future rewards were adjusted. The current prediction function may be too simple. With a better-designed prediction function and a suitable step size value, the performance of the Temporal Difference reward scheme could be further improved and widely used in any Hyper-heuristic framework.

In the cooperative Hyper-heuristic framework, the agents were tested with initial given static weights/priorities in this thesis. Although the test result suggest that the random handling approach used by the mediator performs better. Therefore, dynamic priorities could’ve be a possible approach to improve priority based handling scheme and should be the focus of future research. The exam agents could be given a suitable initial priority and updated during the process. This approach could enable the mediator of the cooperative

framework to learn. Multi-agent Reinforcement Learning could be one of the choices if the cooperative framework consists of agents with different objectives. Also, the exam agents could be a collection of Hyper-heuristic agents as well as other non-Hyper-Heuristic agents. In this way, the strength of the Hyper-heuristic approaches could be with that of non-Hyper-Heuristic approaches to further improve the cooperative search framework.

APPENDIX A. FILES FORMATTING FOR ITC2007

A.1 Input formatting

For ITC2007 benchmarks, the input formats are given on the International Timetabling Competition website (McCollum and McMullan, 2007a). The input format of the problem is shown in Table 56 below.

Table 56:Format of Input Files ITC2007

Format	Content
Number of Exams	The number of exams to be scheduled. For example, [Exam: 3286]
Number of Periods	The duration of one exam. For example, [Period: 45]
The sequence of lines detailing Period Dates, Times, Durations associated Penalty	A set of information about the exam. For example, [29:04:2018, 14:00:00, 180,0] where 29:04:2018 means the period dates of the exam. 14:00:00 means the time of the exams. 180 means the duration of the exams. 0 means the associated penalty.
Number of Rooms	The number of rooms required for the exam. For example, [Rooms: 25]
The sequence of lines detailing room capacity and associated penalty	For example, [300, 7] Where 300 is the capacity of the room. 7 is the penalty.
Period-Related Hard Constraints	It begins with [PeriodHardConstraints] and then the condition of the data. For example, [1, EXCLUSION,3], meaning exam 1 and exam 3 must not be scheduled in the same period.

Room-Related Hard Constraints	It begins with [RoomHardConstraints] and then the condition of the room. For example, [2, ROOM_EXCLUSION], meaning exam 2 must be scheduled in a room.
Institutional Model Weightings	It begins with [InstitutionalWeightings] and then the condition of soft constraints. For example, [THREEINADAY,5]

A.2 Output Formatting

For ITC2007 benchmarks, the output formats are given on the International Timetabling Competition website (McCollum and McMullan, 2007b). The output format of the problem is shown in the Table 57 below.

Table 57:Format of Output Files ITC2007

Number	Format
1	The following information relating to an exam should be provided: The timeslot number and the room number. The timeslot number is an integer between 0 and 44 representing the timeslots allocated to the event. The room number is the number of the room assigned to the event. Remember, rooms are numbered in the order shown in the problem file, starting at zero.
2	One line should describe each exam.
3	The exams should be in the same order as that given in the input file.

REFERENCES

ABRAMSON, D., KRISHNAMOORTHY, M. and DANG, H., 1999. Simulated annealing cooling schedules for the school timetabling problem. *Asia Pacific Journal of Operational Research,* **16**, pp. 1-22.

AHMADI, S., BARONE, R., CHENG, P., COWLING, P. and MCCOLLUM, B., 2003. Perturbation based variable neighbourhood search in heuristic space for examination timetabling problem. *Proceedings of multidisciplinary international scheduling: theory and applications (MISTA 2003),* pp. 13-16.

AICKELIN, U., BURKE, E.K. and LI, J., 2009. An evolutionary squeaky wheel optimization approach to personnel scheduling. *IEEE Transactions on evolutionary computation,* **13**(2), pp. 433-443.

AICKELIN, U. and LI, J., 2007. An estimation of distribution algorithm for nurse scheduling. *Annals of Operations Research,* **155**(1), pp. 289-309.

ASMUNI, H., BURKE, E.K., GARIBALDI, J.M. and MCCOLLUM, B., 2004. Fuzzy multiple heuristic orderings for examination timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 2004, Springer, pp. 334-353.

ASTA, S. and ÖZCAN, E., 2014. An apprenticeship learning hyper-heuristic for vehicle routing in HyFlex, *Evolving and Autonomous Learning Systems (EALS), 2014 IEEE Symposium on* 2014, IEEE, pp. 65-72.

BACK, T., 1996. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms.* Oxford university press.

BAI, R., 2005. An investigation of novel approaches for optimising retail shelf space allocation. *An investigation of novel approaches for optimising retail shelf space allocation.*

BAI, R., BLAZEWICZ, J., BURKE, E.K., KENDALL, G. and MCCOLLUM, B., 2007. A simulated annealing hyper-heuristic methodology for flexible decision support. *School of CSiT, University of Nottingham, UK, Tech.Ren.*

BAI, R. and KENDALL, G., 2005. An investigation of automated planograms using a simulated annealing based hyper-heuristic. *Metaheuristics: Progress as real problem solvers.* Springer, pp. 87-108.

BALCH, T., 1997. Integrating rl and behavior-based control for soccer.

BARBULESCU, L., WATSON, J., WHITLEY, L.D. and HOWE, A.E., 2004. Scheduling space–ground communications for the air force satellite control network. *Journal of Scheduling,* **7**(1), pp. 7-34.

BARTO, A.G., SUTTON, R.S. and ANDERSON, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics,* (5), pp. 834-846.

BATTITI, R. and MASCIA, F., 2007. An algorithm portfolio for the sub-graph isomorphism problem. *Engineering Stochastic Local Search Algorithms. Designing, Implementing and Analyzing Effective Heuristics.* Springer, pp. 106-120.

BELLMAN, R. and KALABA, R., Dynamic programming and modern control theory. 1965. *Academic, New York.*

BERENJI, H.R. and KHEDKAR, P., 1992. Learning and tuning fuzzy logic controllers through reinforcements. *IEEE Transactions on Neural Networks,* **3**(5), pp. 724-740.

BILGIN, B., ÖZCAN, E. and KORKMAZ, E.E., 2006. An experimental study on hyper-heuristics and exam timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 2006, Springer, pp. 394-412.

BOIZUMAULT, P., DELON, Y. and PÉRIDY, L., 1995. A CLP approach for examination planning. *Constraint Processing.* Springer, pp. 85-101.

BOIZUMAULT, P., GUÉRET, C. and JUSSIEN, N., 1994. Efficient labelling and constraint relaxation for solving time tabling problems, *International Logic Programming Symposium-Workshop on Constraint Languages and their use in Problem Modelling* 1994, pp. 116-130.

BONDY, J.A. and MURTY, U.S.R., 1976. *Graph theory with applications.* Citeseer.

BURKE, E.K., ELLIMAN, D.G. and WEARE, R.F., 1995. The automation of the timetabling process in higher education. *Journal of Educational Technology Systems,* **23**(4), pp. 353-362.

BURKE, E.K., CURTOIS, T., HYDE, M., KENDALL, G., OCHOA, G., PETROVIC, S. and VÁZQUEZ-RODRÍGUEZ, J.A., 2009. Hyflex: A flexible framework for the design and analysis of hyper-heuristics, *Multidisciplinary International Scheduling Conference (MISTA 2009), Dublin, Ireland* 2009, pp. 790-797.

BURKE, E.K., GENDREAU, M., OCHOA, G. and WALKER, J.D., 2011. Adaptive iterated local search for cross-domain optimisation, *Proceedings of the 13th annual conference on Genetic and evolutionary computation* 2011, ACM, pp. 1987-1994.

BURKE, E.K., HYDE, M.R. and KENDALL, G., 2011. A squeaky wheel optimisation methodology for two-dimensional strip packing. *Computers & Operations Research,* **38**(7), pp. 1035-1044.

BURKE, E.K., HYDE, M., KENDALL, G., OCHOA, G., ÖZCAN, E. and WOODWARD, J.R., 2010. A classification of hyper-heuristic approaches. *Handbook of metaheuristics.* Springer, pp. 449-468.

BURKE, E.K., MCCOLLUM, B., MCMULLAN, P. and QU, R., 2006. Examination timetabling: a new formulation, *Proceedings of the 6th International Conference on the Practice and Theory of Automated Timetabling, Brno* 2006, pp. 373-375.

BURKE, E.K., MCCOLLUM, B., MEISELS, A., PETROVIC, S. and QU, R., 2007. A graph-based hyper-heuristic for educational timetabling problems. *European Journal of Operational Research,* **176**(1), pp. 177-192.

BURKE, E.K. and NEWALL, J.P., 2004. Solving examination timetabling problems through adaption of heuristic orderings. *Annals of operations Research,* **129**(1-4), pp. 107-134.

BURKE, E.K. and NEWALL, J.P., 1999. A multistage evolutionary algorithm for the timetable problem. *IEEE transactions on evolutionary computation,* **3**(1), pp. 63-74.

BURKE, E.K., NEWALL, J.P. and WEARE, R.F., 1998. Initialization strategies and diversity in evolutionary timetabling. *Evolutionary computation,* **6**(1), pp. 81-103.

BURKE, E.K., NEWALL, J.P. and WEARE, R.F., 1995. A memetic algorithm for university exam timetabling, *international conference on the practice and theory of automated timetabling* 1995, Springer, pp. 241-250.

BURKE, E., BYKOV, Y., NEWALL, J. and PETROVIC, S., 2004. A time-predefined local search approach to exam timetabling problems. *Iie Transactions,* **36**(6), pp. 509-528.

BURKE, E., DROR, M., PETROVIC, S. and QU, R., 2005. Hybrid graph heuristics within a hyper-heuristic approach to exam timetabling problems. *The next wave in computing, optimization, and decision technologies.* Springer, pp. 79-91.

BURKE, E., JACKSON, K., KINGSTON, J.H. and WEARE, R., 1997. Automated university timetabling: The state of the art. *The computer journal,* **40**(9), pp. 565-571.

BURKE, E., KENDALL, G., NEWALL, J., HART, E., ROSS, P. and SCHULENBURG, S., 2003. Hyper-heuristics: An emerging direction in modern search technology. *Handbook of metaheuristics.* Springer, pp. 457-474.

CARTER, M.W., 2001. Scheduling and sequencing Timetabling. *Encyclopedia of operations research and management science.* Springer, pp. 833-836.

CARTER, M.W. and LAPORTE, G., 1995. Recent developments in practical examination timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 1-21.

CARTER, M.W., LAPORTE, G. and CHINNECK, J.W., 1994. A general examination scheduling system. *Interfaces,* **24**(3), pp. 109-120.

CARTER, M.W., LAPORTE, G. and LEE, S.Y., 1996. Examination timetabling: Algorithmic strategies and applications. *Journal of the Operational Research Society,* **47**(3), pp. 373-383.

CHAKHLEVITCH, K. and COWLING, P., 2008. Hyperheuristics: recent developments. *Adaptive and multilevel metaheuristics.* Springer, pp. 3-29.

CHENG, C., KANG, L., LEUNG, N. and WHITE, G.M., 1995. Investigations of a constraint logic programming approach to university timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 112-129.

COOPER, T.B. and KINGSTON, J.H., 1995. The complexity of timetable construction problems, *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 281-295.

CORNE, D., ROSS, P. and FANG, H., 1994. Evolutionary timetabling: Practice, prospects and work in progress, *In Proceedings of the UK Planning and Scheduling SIG Workshop, Strathclyde* 1994.

CÔTÉ, P., WONG, T. and SABOURIN, R., 2004. A hybrid multi-objective evolutionary algorithm for the uncapacitated exam proximity problem, *International Conference on the Practice and Theory of Automated Timetabling* 2004, Springer, pp. 294-312.

COWLING, P., KENDALL, G. and HAN, L., 2002. An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem, *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on* 2002, IEEE, pp. 1185-1190.

COWLING, P., KENDALL, G. and SOUBEIGA, E., 2002. Hyperheuristics: A robust optimisation method applied to nurse scheduling, *International Conference on Parallel Problem Solving from Nature* 2002, Springer, pp. 851-860.

COWLING, P., KENDALL, G. and SOUBEIGA, E., 2001. A parameter-free hyperheuristic for scheduling a sales summit, *Proceedings of the 4th metaheuristic international conference* 2001, Citeseer, pp. 127-131.

CRITES, R.H. and BARTO, A.G., 1996. Improving elevator performance using reinforcement learning, *Advances in neural information processing systems* 1996, pp. 1017-1023.

DI GASPERO, L., MCCOLLUM, B. and SCHAERF, A., 2007. The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3). *The second international timetabling competition (ITC-2007): Curriculum-based course timetabling (track 3).*

DI GASPERO, L. and SCHAERF, A., 2002. Multi-neighbourhood local search with application to course timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 2002, Springer, pp. 262-275.

DI GASPERO, L. and SCHAERF, A., 2000. Tabu search techniques for examination timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 2000, Springer, pp. 104-117.

DIMOPOULOU, M. and MILIOTIS, P., 2004. An automated university course timetabling system developed in a distributed environment: A case study. *European Journal of Operational Research,* **153**(1), pp. 136-147.

DORIGO, M. and BIRATTARI, M., 2011. Ant colony optimization. *Encyclopedia of machine learning.* Springer, pp. 36-39.

DOUCET, A., DE FREITAS, N. and GORDON, N., 2001. An introduction to sequential Monte Carlo methods. *Sequential Monte Carlo methods in practice.* Springer, pp. 3-14.

DR BARRY MCCOLLUM DR PAUL MCMULLAN, 2007a-last update, Input Format for ITC2007 Track 1: Exam Timetabling, Available: http://www.cs.qub.ac.uk/itc2007/examtrack/exam_track_index_files/Inputformat.htm.

DR BARRY MCCOLLUM DR PAUL MCMULLAN, 2007b. Output Format for ITC2007 Track 1: Exam Timetabling.

ERBEN, W., 2000. A grouping genetic algorithm for graph colouring and exam timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 2000, Springer, pp. 132-156.

EVEN-DAR, E. and MANSOUR, Y., 2003. Learning rates for Q-learning. *Journal of Machine Learning Research,* **5**(Dec), pp. 1-25.

FENG, G. and LAU, H.C., 2008. Efficient algorithms for machine scheduling problems with earliness and tardiness penalties. *Annals of Operations Research,* **159**(1), pp. 83-95.

FEO, T.A. and RESENDE, M.G., 1995. Greedy randomized adaptive search procedures. *Journal of Global Optimization,* **6**(2), pp. 109-133.

FU, Z., LI, Y., LIM, A. and RODRIGUES, B., 2007. Port space allocation with a time dimension. *Journal of the Operational Research Society,* **58**(6), pp. 797-807.

GINSBERG, M.L., 2001. GIB: Imperfect information in a computationally challenging game. *Journal of Artificial Intelligence Research,* **14**, pp. 303-358.

GLOVER, F., 1989. Tabu search—part I. *ORSA Journal on computing,* **1**(3), pp. 190-206.

GUÉRET, C., JUSSIEN, N., BOIZUMAULT, P. and PRINS, C., 1995. Building university timetables using constraint logic programming, *International conference on the practice and theory of automated timetabling* 1995, Springer, pp. 130-145.

HAJIAGHAYI, M., KIRKPATRICK, B., WANG, L. and BOUCHARD-CÔTÉ, A., 2014. Efficient continuous-time Markov chain estimation, *International Conference on Machine Learning* 2014, pp. 638-646.

HANSEN, P., MLADENOVIC, N. and PÉREZ, J.A.M., 2003. Variable neighbourhood search. *Inteligencia Artificial, Revista Iberoamericana de Inteligencia Artificial,* **19**, pp. 77-92.

HERTZ, A., 1992. Finding a feasible course schedule using tabu search. *Discrete Applied Mathematics,* **35**(3), pp. 255-270.

HERTZ, A., 1991. Tabu search for large scale timetabling problems. *European Journal of Operational Research,* **54**(1), pp. 39-47.

HSIAO, P., CHIANG, T. and FU, L., 2012. A vns-based hyper-heuristic with adaptive computational budget of local search, *Evolutionary Computation (CEC), 2012 IEEE Congress on* 2012, IEEE, pp. 1-8.

HU, J. and WELLMAN, M.P., 2003. Nash Q-learning for general-sum stochastic games. *Journal of machine learning research,* **4**(Nov), pp. 1039-1069.

INGBER, L., 1993. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling,* **18**(11), pp. 29-57.

JOSLIN, D.E. and CLEMENTS, D.P., 1999. Squeaky wheel optimization. *Journal of Artificial Intelligence Research,* **10**, pp. 353-373.

KALENDER, M., KHEIRI, A., ÖZCAN, E. and BURKE, E.K., 2013. A greedy gradient-simulated annealing selection hyper-heuristic. *Soft Computing,* **17**(12), pp. 2279-2292.

KAMEI, K. and ISHIKAWA, M., 2004. Determination of the optimal values of parameters in reinforcement learning for mobile robot navigation by a genetic algorithm, *International Congress Series* 2004, Elsevier, pp. 193-196.

KANG, L. and WHITE, G.M., 1992. A logic approach to the resolution of constraints in timetabling. *European Journal of Operational Research,* **61**(3), pp. 306-317.

KAPLANSKY, E., KENDALL, G., MEISELS, A. and HUSSIN, N., 2004. Distributed examination timetabling, *Proc. of the 5th International Conference of the Practice and Theory of Automated Timetabling (PATAT), Pittsburg, USA* 2004, pp. 511-516.

KAPLANSKY, E. and MEISELS, A., 2004. Negotiation among scheduling agents for distributed timetabling, *Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling, Pittsburgh* 2004, Citeseer, pp. 517-520.

KELIY, J.P., 1996. META-HEURISTICS: Theory & Applications.

KELLER, R.E. and POLI, R., 2007. Linear genetic programming of parsimonious metaheuristics, *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on* 2007, IEEE, pp. 4508-4515.

KENDALL, G. and HUSSIN, N.M., 2004. A tabu search hyper-heuristic approach to the examination timetabling problem at the MARA university of technology, *International Conference on the Practice and Theory of Automated Timetabling* 2004, Springer, pp. 270-293.

KHEIRI, A. and ÖZCAN, E., 2016. An iterated multi-stage selection hyper-heuristic. *European Journal of Operational Research,* **250**(1), pp. 77-90.

KLOPF, A.H., 1972. Brain function and adaptive systems: a heterostatic theory. *Brain function and adaptive systems: a heterostatic theory.*

LI, J. and KWAN, R.S., 2003. A fuzzy genetic algorithm for driver scheduling. *European Journal of Operational Research,* **147**(2), pp. 334-344.

LITTMAN, M.L., 2001. Friend-or-foe Q-learning in general-sum games, *ICML* 2001, pp. 322-328.

LITTMAN, M.L., 1994. Markov games as a framework for multi-agent reinforcement learning. *Machine Learning Proceedings 1994.* Elsevier, pp. 157-163.

LOURENÇO, H.R., MARTIN, O.C. and STÜTZLE, T., 2003. Iterated local search. *Handbook of metaheuristics.* Springer, pp. 320-353.

MAES, P. and BROOKS, R.A., 1990. Learning to Coordinate Behaviors. *AAAI* 1990, pp. 796-802.

MAKAR, R., MAHADEVAN, S. and GHAVAMZADEH, M., 2001. Hierarchical multi-agent reinforcement learning, *Proceedings of the fifth international conference on Autonomous agents* 2001, ACM, pp. 246-253.

MALAGUTI, E. and TOTH, P., 2008. An evolutionary approach for bandwidth multicoloring problems. *European Journal of Operational Research,* **189**(3), pp. 638-651.

MARQUES-SILVA, J.P. and SAKALLAH, K.A., 1999. GRASP: A search algorithm for propositional satisfiability. *IEEE Transactions on Computers,* **48**(5), pp. 506-521.

MCCOLLUM, B., MCMULLAN, P., BURKE, E.K., PARKES, A.J. and QU, R., 2007. The second international timetabling competition: Examination timetabling track. *The second international timetabling competition: Examination timetabling track,* .

MERLOT, L.T., BOLAND, N., HUGHES, B.D. and STUCKEY, P.J., 2002. A hybrid algorithm for the examination timetabling problem, *International Conference on the Practice and Theory of Automated Timetabling* 2002, Springer, pp. 207-231.

METROPOLIS, N., ROSENBLUTH, A., ROSENBLUTH, M., TELLER, A. and TELLER, E., 1953. Simulated annealing. *Journal of Chemical Physics,* **21**, pp. 1087-1092.

MICHAUD, F. and MATARIC, M.J., 1998. Learning from history for adaptive mobile robot control, *Intelligent Robots and Systems, 1998. Proceedings., 1998 IEEE/RSJ International Conference on* 1998, IEEE, pp. 1865-1870.

MISIR, M., VERBEECK, K., DE CAUSMAECKER, P. and BERGHE, G.V., 2012. An intelligent hyper-heuristic framework for chesc 2011. *Learning and Intelligent Optimization.* Springer, pp. 461-466.

MLADENOVIĆ, N. and HANSEN, P., 1997. Variable neighborhood search. *Computers & Operations Research,* **24**(11), pp. 1097-1100.

MOSCATO, P., COTTA, C. and MENDES, A., 2004. Memetic algorithms. *New optimization techniques in engineering.* Springer, pp. 53-85.

NAHAR, S., SAHNI, S. and SHRAGOWITZ, E., 1986. Simulated annealing and combinatorial optimization, *Proceedings of the 23rd ACM/IEEE design automation conference* 1986, IEEE Press, pp. 293-299.

OUELHADJ, D. and PETROVIC, S., 2010. A cooperative hyper-heuristic search framework. *Journal of Heuristics,* **16**(6), pp. 835-857.

OUELHADJ, D. and PETROVIC, S., 2008. A cooperative distributed hyper-heuristic framework for scheduling, *Systems, Man and Cybernetics, 2008. SMC 2008. IEEE International Conference on* 2008, IEEE, pp. 2560-2565.

ÖZCAN, E., BILGIN, B. and KORKMAZ, E.E., 2008. A comprehensive analysis of hyper-heuristics. *Intelligent Data Analysis,* **12**(1), pp. 3-23.

ÖZCAN, E. and KHEIRI, A., 2011. A hyper-heuristic based on random gradient, greedy and dominance. *Computer and Information Sciences II.* Springer, pp. 557-563.

ÖZCAN, E., MISIR, M., OCHOA, G. and BURKE, E.K., 2012. A Reinforcement Learning: Great-Deluge Hyper-Heuristic for Examination Timetabling. *Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends.* IGI Global, pp. 34-55.

PAECHTER, B., CUMMING, A., NORMAN, M.G. and LUCHIAN, H., 1995. Extensions to a memetic timetabling system, *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 251-265.

b.PAQUETE, L. and STUTZLE, T., 2002. Empirical analysis of tabu search for the lexicographic optimization of the examination timetabling problem, *2002). Proceedings of the 4th International Conference on Practice and Theory of Automated Timetabling. 21st-23rd August* 2002, pp. 413-420.

a.PAQUETE, L. and STÜTZLE, T., 2002. An experimental investigation of iterated local search for coloring graphs, *Workshops on Applications of Evolutionary Computation* 2002, Springer, pp. 122-131.

PENG, J. and WILLIAMS, R.J., 1994. Incremental multi-step Q-learning. *Machine Learning Proceedings 1994.* Elsevier, pp. 226-232.

PETROVIC, S. and BURKE, E.K., 2004. University Timetabling. *University Timetabling.*

QU, R., BURKE, E.K., MCCOLLUM, B., MERLOT, L.T. and LEE, S.Y., 2006. A survey of search methodologies and automated approaches for examination timetabling. *Computer Science Technical Report No. NOTTCS-TR-2006-4, UK.*

QU, R., BURKE, E.K. and MCCOLLUM, B., 2009. Adaptive automated construction of hybrid heuristics for exam timetabling and graph colouring problems. *European Journal of Operational Research,* **198**(2), pp. 392-404.

QU, R., BURKE, E.K., MCCOLLUM, B., MERLOT, L.T. and LEE, S.Y., 2009. A survey of search methodologies and automated system development for examination timetabling. *Journal of scheduling,* **12**(1), pp. 55-89.

REEVES, C.R., 1995. *Modern heuristic techniques for combinatorial problems. Advanced topics in computer science.* Mc Graw-Hill.

ROSS, P., CORNE, D. and TERASHIMA-MARÍN, H., 1995. The phase-transition niche for evolutionary algorithms in timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 309-324.

ROSS, P., HART, E. and CORNE, D., 1997. Some observations about GA-based exam timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 1997, Springer, pp. 115-129.

ROSS, P., MARÍN-BLÁZQUEZ, J.G. and HART, E., 2004. Hyper-heuristics applied to class and exam timetabling problems, *Evolutionary Computation, 2004. CEC2004. Congress on* 2004, IEEE, pp. 1691-1698.

SABAR, N.R., AYOB, M., QU, R. and KENDALL, G., 2012. A graph coloring constructive hyper-heuristic for examination timetabling problems. *Applied Intelligence,* **37**(1), pp. 1-11.

SAMUEL, A.L., 1959. Some studies in machine learning using the game of checkers. *IBM Journal of research and development,* **3**(3), pp. 210-229.

SANTAMARIA, J.C. and RAM, A., 1997. Multi strategy learning of adaptive reactive controllers. *Multistrategy Learning of Adaptive Reactive Controllers.*

SCHAERF, A. and DI GASPERO, L., 2001. Local search techniques for educational timetabling problems, *Proceedings of the 6th International Symposium on Operational Research (SOR-01), Preddvor, Slovenia* 2001, pp. 13-23.

SÖRENSEN, K. and GLOVER, F.W., 2013. Metaheuristics. *Encyclopedia of operations research and management science.* Springer, pp. 960-970.

SOUBEIGA, E., 2003. Development and application of hyper-heuristics to personnel scheduling. *Development and application of hyperheuristics to personnel scheduling, .*

SUTTON, R.S., 1988. Learning to predict by the methods of temporal differences. *Machine Learning,* **3**(1), pp. 9-44.

SUTTON, R.S. and BARTO, A.G., 1998. *Reinforcement learning: An introduction.* MIT press.

TAN, M., 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents, *Proceedings of the tenth international conference on machine learning* 1993, pp. 330-337.

TERASHIMA-MARIN, H., ROSS, P. and VALENZUELA-RENDON, M., 1999. Clique-based crossover for solving the timetabling problem with GAs, *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on* 1999, IEEE, pp. 1200-1206.

THOMPSON, J.M. and DOWSLAND, K.A., 1996. Variants of simulated annealing for the examination timetabling problem. *Annals of Operations research,* **63**(1), pp. 105-128.

THOMPSON, J. and DOWSLAND, K.A., 1995. General cooling schedules for a simulated annealing based timetabling system, *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 345-363.

ÜLKER, Ö, ÖZCAN, E. and KORKMAZ, E.E., 2006. Linear linkage encoding in grouping problems: applications on graph coloring and timetabling, *International Conference on the Practice and Theory of Automated Timetabling* 2006, Springer, pp. 347-363.

VAN BREEDAM, A., 2001. Comparing descent heuristics and metaheuristics for the vehicle routing problem. *Computers & Operations Research,* **28**(4), pp. 289-315.

WATKINS, C.J. and DAYAN, P., 1992. Q-learning. *Machine Learning,* **8**(3-4), pp. 279-292.

WEARE, R., BURKE, E. and ELLIMAN, D., 1995. A hybrid genetic algorithm for highly constrained timetabling problems. *Department of Computer Science.*

WELSH, D.J. and POWELL, M.B., 1967. An upper bound for the chromatic number of a graph and its application to timetabling problems. *The Computer Journal,* **10**(1), pp. 85-86.

WHITE, G.M. and XIE, B.S., 2000. Examination timetables and tabu search with longer-term memory, *International Conference on the Practice and Theory of Automated Timetabling* 2000, Springer, pp. 85-103.

WHITLEY, D., 1994. A genetic algorithm tutorial. *Statistics and computing,* **4**(2), pp. 65-85.

WIDROW, B., GUPTA, N.K. and MAITRA, S., 1973. Punish/reward: Learning with a critic in adaptive threshold systems. *IEEE transactions on systems, man, and cybernetics,* (5), pp. 455-465.

WOLPERT, D.H. and MACREADY, W.G., 1997. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation,* **1**(1), pp. 67-82.

WONG, T., CÔTÉ, P. and GELY, P., 2002. Final exam timetabling: a practical approach, *IEEE Canadian conference on electrical and computer engineering (CCECE 2002)* 2002, Citeseer, pp. 726-731.

WREN, A., 1995. Scheduling, timetabling and rostering—a special relationship? *International Conference on the Practice and Theory of Automated Timetabling* 1995, Springer, pp. 46-75.

XU, J., CHIU, S.Y. and GLOVER, F., 1998. Fine-tuning a Tabu Search Algorithm with Statistical Tests. *International Transactions in Operational Research,* **5**(3), pp. 233-244.

YASUNOBU, S. and MATSUBARA, T., 2003. Fuzzy target acquired by reinforcement learning for parking control, *SICE 2003 Annual Conference* 2003, IEEE, pp. 1242-1247.

ZAMSTEIN, L.M., ARROYO, A.A., SCHWARTZ, E.M., KEEN, S., SUTTON, B.C. and GANDHI, G., 2006. Koolio: Path planning using reinforcement learning on a real robot platform, *FCRAR, 19th Florida Conference on Recent Advances in Robotics* 2006, pp. 25-26.

ÖZCAN, E. and ALPAY A., 2002. Timetabling using a steady state genetiv algorithm. In PATAT, Vol.2, pp. 104-107

BURKE, E.K., KENDALL, G. and SOUBEIGA, E., 2003. A tabu-search hyperheuristic for timetabling and rostering. *Journal of heuristics*, 9(6), pp.451-470.

ARAI, S., SYCARA, K. and PAYNE, T.R., 2000, August. Experience-based reinforcement learning to acquire effective behavior in a multi-agent domain. *In Pacific Rim International Conference on Artificial Intelligence*, pp. 125-135. Springer, Berlin, Heidelberg.

BARTO, A.G., SUTTON, R.S. and ANDERSON, C.W., 1983. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, (5), pp.834-846.

TESAURO, G., 1995. Temporal difference learning and TD-Gammon. *Communications of the ACM*, 38(3), pp.58-68.

UCHIBE, E., ASADA, M., NODA, S., TAKAHASHI, Y. and HOSODA, K., 1996. Vision-Based Reinforcement Learning for RoboCup: Towards Real Robot Competition. *In Proc. of IROS* (Vol. 96).

BURKE, E.K., Kendall, G., MISIR, M. and ÖZCAN, E., 2012. Monte Carlo hyper-heuristics for examination timetabling. *Annals of Operations Research*, 196(1), pp.73-90.

MALONE, T.W. and CROWSTON, K., 1994. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR),* 26(1), pp.87-119.

ELS, R. and PILLAY, N., 2010, December. An evolutionary algorithm hyper-heuristic for producing feasible timetables for the curriculum based university course timetabling problem. *In Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress*, pp. 460-466. IEEE.

ANWAR, K., KHADER, A.T., AL-BETAR, M.A. and AWADALLAH, M.A., 2013, March. Harmony search-based hyper-heuristic for examination timetabling. In Signal Processing and its Applications (CSPA). *2013 IEEE 9th International Colloquium, Vol. 9*, pp. 176-181.

GOGOS, C., ALEFRAGIS, P. and HOUSOS, E., 2012. An improved multi-staged algorithmic process for the solution of the examination timetabling problem. *Annals of Operations Research*, 194(1), pp.203-221.

RAHMAN, S.A., BARGIELA, A., BURKE, E.K., ÖZCAN, E., MZCOLLUM, B. and MCMULLAN, P., 2014. Adaptive linear combination of heuristic orderings in constructing examination timetables. *European Journal of Operational Research*, 232(2), pp.287-297.

MULLER, T., 2009. ITC2007 solver description: A hybrid approach. *Annals of Operations Research*, 172(1), p.429.

BURKE, E.K., QU, R. and SOGHIER, A., 2014. Adaptive selection of heuristics for improving exam timetables. *Annals of Operations Research*, 218(1), pp.129-145.

HAMILTON-BRYCE, R., MCMULLAN, P. and MCCLLOUM, B., 2014, August. Directed selection using reinforcement learning for the examination timetabling problem. In PATAT'14 *Proceedings of the 10th International Conference on the Practice and Theory of Automated Timetabling*.